

DTIC FILE COPY

AD-A229 932

METHODOLOGY DEVELOPMENT FOR THE VERIFICATION AND  
VALIDATION OF FLIGHT CRITICAL SYSTEMS SOFTWARE



Ronald L. Braet

Frontier Technology, Inc.  
4141 Colonel Glenn Highway  
Beavercreek, OH 45431

October 1990

Final Report for Period Dec 89 - Aug 90

DTIC  
ELECTE  
DEC 04 1990  
S E D  
Cx

Approved for public release; distribution unlimited


FLIGHT DYNAMICS LABORATORY  
WRIGHT RESEARCH DEVELOPMENT CENTER  
AIR FORCE SYSTEMS COMMAND  
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6553

NOTICE

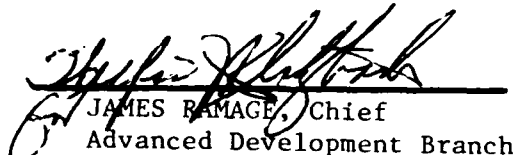
When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.




ANTHONY P. DeTHOMAS  
Project Engineer  
Advanced Development Branch  
Flight Control Division



JAMES RAMAGE, Chief  
Advanced Development Branch  
Flight Control Division

FOR THE COMMANDER



H. MAX DAVIS, Assistant for  
Research and Technology  
Flight Control Division  
Flight Dynamics Laboratory

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/FIGX, WPAFB, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS N/A		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) FTI-9042-001			5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-3067		
6a. NAME OF PERFORMING ORGANIZATION FRONTIER TECHNOLOGY, INC.		6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Flight Dynamics Laboratory (WRDC/FIGX) Wright Research Development Center		
6c. ADDRESS (City, State, and ZIP Code) 4141 Colonel Glenn Highway Beavercreek, OH 45431			7b. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6553		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Dept of Air Force ASD/PMRNB		8b. OFFICE SYMBOL (If applicable) ASD/PMRNB	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-89-C-3610		
8c. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6503			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO. 65502F	PROJECT NO. 3005	TASK NO. 40
11. TITLE (Include Security Classification) METHODOLOGY DEVELOPMENT FOR THE VERIFICATION AND VALIDATION OF FLIGHT CRITICAL SYSTEMS SOFTWARE					
12. PERSONAL AUTHOR(S) BRAET, RONALD L.					
13a. TYPE OF REPORT FINAL		13b. TIME COVERED FROM 12/12/89 to 8/12/90		14. DATE OF REPORT (Year, Month, Day) 1990 October	
15. PAGE COUNT 107					
16. SUPPLEMENTARY NOTATION This is a Small Business Innovative Research Program, Phase I report.					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) software flight critical systems, CASE, SOF, software verification and validation, s/w development		
FIELD	GROUP	SUB-GROUP			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The results of this conceptual design study have shown that there is a growing need to develop a methodology by which flight critical systems software can be verified and validated for performance and safety impacts. The nature of the evolving technology and its application to FCS software verification and validation requirements has current V&V methods lagging behind design methods and tools. It is recognized throughout government and industry that FCS software V&V requires knowledgeable and skilled individuals utilizing proper tools and techniques to successfully complete the V&V effort in a timely manner. This report provides an overview of the development process of flight critical systems and the roles of verification and validation which go hand-in-hand with the development process. It provides a conceptual design for a computer aided environment to perform FCS software verification and validation.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/DUNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL ANTHONY DETHOMAS			22b. TELEPHONE (Include Area Code) (513) 255-8474		22c. OFFICE SYMBOL WRDC/FIGX

## TABLE OF CONTENTS

Section	Page
I. INTRODUCTION	
1.1 Purpose.....	1
1.2 Scope.....	1
1.3 Document Organization.....	3
II. EXECUTIVE SUMMARY	
2.1 Purpose Of The Work.....	4
2.2 Overview of Technical Approach.....	6
2.2.1 Step 1 - Requirements Analysis.....	7
2.2.2 Step 2 - Data Collection.....	8
2.2.3 Step 3 Verification and Validation Concept Design.....	9
2.3 Summary of the Results.....	11
2.3.1 Results of FCS Software V&V Requirements Definition and Data Collection.....	11
2.3.2 Conceptual Design for a V&V Methodology for FCS Software.....	17
2.3.3 CAV-ES FCS V&V Capabilities.....	21
2.4 Potential Applications Of The Effort.....	25
III. TECHNICAL DISCUSSIONS	
3.1 Development Phases For Flight Critical Systems.....	28
3.1.1 Overview.....	28
3.1.2 System Requirements/Design Phase.....	30
3.1.3 Subsystem Requirements/Design Phase.....	30
3.1.4 Software Requirements/Hardware Specification Phase.....	31
3.1.5 Basic Software Design Phase.....	31
3.1.6 Detailed Software Design Phase.....	32
3.1.7 Module Coding/Test Phase.....	32
3.1.8 Software & Software/Hardware Integration Phase.....	33
3.1.9 Subsystem Integration Phase.....	33
3.1.10 System Integration Phase.....	33
3.1.11 Flight Critical System Engineering.....	34
3.1.12 Flight Test Phase.....	35
3.2 Verification and Validation Of Flight Critical Systems Software.....	35
3.2.1 Overview.....	35
3.2.2 FCS Software Requirements Analysis.....	36
3.2.3 FCS Software Design Analysis.....	37
3.2.4 Code Analysis.....	38
3.2.5 Flight Critical Systems Software Test.....	39
3.2.6 FCS Software V&V Tools and Techniques.....	41
3.3 Task 1 Results: Requirements For FCS Software V&V.....	47
3.3.1 General Requirements for Flight Critical Systems.....	47

3.3.2	FCS Design Trends Impact on Software V&V Requirements.....	47
3.3.3	Technology Impacts on FCS Software V&V Requirements.....	50
3.4	Task 2 Results: Data Collection.....	50
3.5	Task 3 Results: Development of a FCS V&V Methodology.....	59
3.5.1	Technical Approach.....	59
3.5.2	Computer Aided V&V Engineering System Design Overview.....	61
3.5.3	CAVES FCS V&V Capabilities.....	68
3.5.3.1	Aircraft Flight Critical Systems Analysis.....	68
3.5.3.2	FCS Software Design Verification.....	74
3.5.3.3	FCS Software Code Verification.....	78
3.5.3.4	Stand Alone and Dynamic Subsystem V&V.....	83
3.5.3.5	Integrated System Verification and Validation.....	84
Appendix	Data Gathering Checklist.....	89

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



## SECTION 1.0

### INTRODUCTION

#### 1.1 PURPOSE

This document describes the Phase I SBIR study titled "Methodology Development For Verification and Validation Of Flight Critical Systems Software", Contract No. F33615-89-C-3610. This work was sponsored by the Wright Research and Development Center, Flight Dynamics Laboratory, Flight Control Division at Wright Patterson Air Force Base. The purpose of this effort was to develop an approach for an innovative, integrated verification and validation methodology which focuses on highly coupled flight critical systems software.

#### 1.2 SCOPE

Current trends in applications of advanced control and integration technologies are bringing about the development of on-board systems that are designed to enhance combat effectiveness and survivability in ever increasing hostile combat environments. Flight critical systems (including integrated flight and propulsion control, integrated flight and fire control, self-repairing flight control, vehicle management, pilot-vehicle interface, and flight vehicle sensors) are being controlled and integrated in software. These developments are putting an ever increasing load on the development, verification and validation (V&V) of flight critical systems software. This Phase I SBIR effort has researched and developed an innovative approach for advancing the state of the art in the application of verification and validation methodology. This methodology can be developed into a Computer Aided Verification and Validation Engineering System (CAV<sup>2</sup>ES) hosted in a workstation environment. FTI's approach provides a quantum improvement in the effectiveness and efficiency of the software environment (containing methods, disciplines, documentation, tools and controls) needed for verification and validation of highly reliable, fault tolerant flight critical systems software.

The CAV<sup>2</sup>ES environment offers many advantages over current V&V methods and procedures. These advantages include:

- o A highly automated system that speeds the V&V process by approximately 5 to 1, saving valuable time and money.
- o A workstation environment which provides the user with a powerful computational environment which can utilize many of the more sophisticated FCS design and analysis tools available.
- o An environment easily adaptable to futuristic Air Force systems such as hypersonic air vehicle, control of unmanned aerial vehicles, and even battle management systems.
- o The capability to evaluate designs and design excursions within the V&V environment and transport the evaluations to other phases of V&V efforts.
- o Provides a CASE type working environment for transportation and traceability of V&V requirements, design, and test analysis data.
- o Is oriented specifically to performing software V&V, but is applicable to development efforts.
- o Provides a User Help Guide to specific V&V steps to be performed.
- o Provides the capability to perform rapid prototyping through its tool and interface environment.
- o Can accept inputs from many external systems: (eg., MEAD, TAE, ... ) through common interfaces defined for transporting external inputs.
- o Can be used to aid in providing a flight critical systems engineering function throughout FCS software development cycle.

The unique features of the Computer Aided Verification and Validation Engineering System makes it a very cost effective environment to be used by the flight critical systems analyst to perform the V&V process. These features include:

- o An integrated V&V tool set usable through all development phases.
- o User friendly interfaces; point/click mouse, pull down windows, quick views and transfer of data/comparisons, common functionality between tools, command driven and menu driven human interface.

- o Provides hooks for adding future tools and a means for interfacing and driving real-time simulations.
- o Offers an environment aiding automation of real-time testing.
- o Usable by developers, V&V organizations, and research groups.
- o Is an evolutionary system which accommodates new tools to meet the growing user requirements.

### 1.3 Document Organization

This report contains the following sections:

- o Section 1 is the Introduction.
- o Section 2 is an Executive Summary of the approach taken in this effort and the results of each task.
- o Section 3, Technical Discussions, contains all of the details of the work performed and presents the Computer Aided Verification and Validation Engineering System conceptual design.



## SECTION 2.0

### EXECUTIVE SUMMARY

#### 2.1 PURPOSE OF THE WORK

The applications of digital technology to Flight Critical Systems (FCS) has allowed far reaching advances in air vehicles through vehicle optimization of the air vehicle controllability, performance, safety, and mission effectiveness. Flight critical systems (including integrated flight and propulsion control, integrated flight and fire control, self-repairing flight control, vehicle management, pilot-vehicle interface, and flight vehicle sensors) are being controlled and integrated through software. The spectrum of flight control technology has expanded beyond the classically recognized role of stability and control and flying qualities and now includes technologies covering functional, physical, and pilot-vehicle interfaces which impact aircraft design options, combat effectiveness, and survivability.

As shown in Table 2.1-1, there are a number of problem areas in the verification and validation of flight critical system software. First, implementation of most flight critical systems is being performed through software. This means that an increasing share of the development effort and costs will continue to be driven by software. Thus, there is a very important demand for a well disciplined, efficient software development, verification, and validation methodology. This study specifically addresses verification and validation methodology, but discusses tools and techniques which are equally applicable during development.

Second, not only are these systems being developed with software being a primary implementing factor, but also software has become the means by which these flight critical systems are integrated. Thus, a well disciplined, efficient software V&V methodology is required, more now than ever before to both guide the development and conduct V&V of highly integrated FCS software.

## TABLE 2.1-1 SOFTWARE PROBLEM AREAS IN VERIFICATION AND VALIDATION OF FLIGHT CRITICAL SYSTEMS SOFTWARE

- EXPANDED ROLE OF SOFTWARE IN FLIGHT CRITICAL SYSTEMS  
REQUIRES A WELL DISCIPLINED SOFTWARE V&V METHODOLOGY
- INTEGRATION OF CRITICAL FLIGHT SYSTEMS THROUGH SOFTWARE  
REQUIRES A RELIABLE V&V PROCESS TO ASSURE THAT SAFETY  
OF FLIGHT REQUIREMENTS ARE MET
- SHIFT TO HOLS REQUIRES DEVELOPMENT/UPDATE OF
  - SOFTWARE TOOLS
  - IMPLEMENTATION ENVIRONMENT
  - VERIFICATION AND VALIDATION TECHNIQUES
- FREQUENT CHANGES IN SOFTWARE DEVELOPMENT & DOCUMENTATION  
STANDARDS REQUIRES FLEXIBLE SOFTWARE DEVELOPMENT AND V&V  
APPROACHES THAT CAN BE UPDATED EFFICIENTLY
- INCREASED COMPLEXITY OF INTEGRATED FLIGHT CRITICAL SYSTEMS  
INCREASES THE MAGNITUDE/COST OF SOFTWARE DEVELOPMENT  
AND V&V EFFORTS

V&V3

Third, the shift from assembly language and certain higher-order languages (i.e., JOVIAL, etc.) to Ada as the accepted language for flight critical systems is in process and is impacting current and future software development in flight critical systems. Efforts are under way to improve the performance of the software implementation technology used within Ada compilers to meet flight critical systems requirements. These improvements, when achieved, will enable the cost effective features of Ada to be used and more effectively provide fault tolerant code which is essential for flight critical systems. However, during this transition period, development of software tools, choice

of different implementation strategies and conversion from previous implementation languages to Ada's implementation structure and discipline will put additional burdens in software development and V&V disciplines.

Fourth, changes in government software development standards effect required documentation levels, milestone reporting, etc., have also added to the need of developing a disciplined software development and V&V methodology. Previously used military standards applicable to software development (MIL-STD-483 and MIL-STD-490) have been amended by MIL-STD-2167A. MIL-STD-2167A has a formidable list of documentation requirements on software which are impacting software development efforts on current flight critical systems. Accountability to these software standards and tailoring of these standards to specific applications is yet another area which must be factored into a software development and V&V methodology.

Finally, the evolvement of highly integrated analog/digital and digital/digital flight critical systems which are mechanized to provide redundancy and the capability to reconfigure or provide graceful degradation with failure insertions has complicated the verification testing of such systems. Coupling these mechanizations with interfaces to sensors avionic systems, pilot interfaces, and AI oriented vehicle management systems has made the cost of software development one of the major drivers in system life cycle cost. In this area alone a disciplined software verification and validation methodology addressing all aspects of critical flight software development can reap tremendous savings.

## 2.2 OVERVIEW OF TECHNICAL APPROACH

The technical approach to the development of a methodology for verification and validation of flight critical systems software was carried out in a three step process. The first step, Task 1, was to perform a requirements analysis to identify specific needs to be met in V&V of flight critical systems software. The second task performed in conjunction with the first was a data

collection effort. The third step (Task 3) was the synthesis of a FCS software V&V methodology which would meet the requirements defined in the requirements analysis, Task 1. This process for the Phase I effort is shown in Figure 2.2-1.

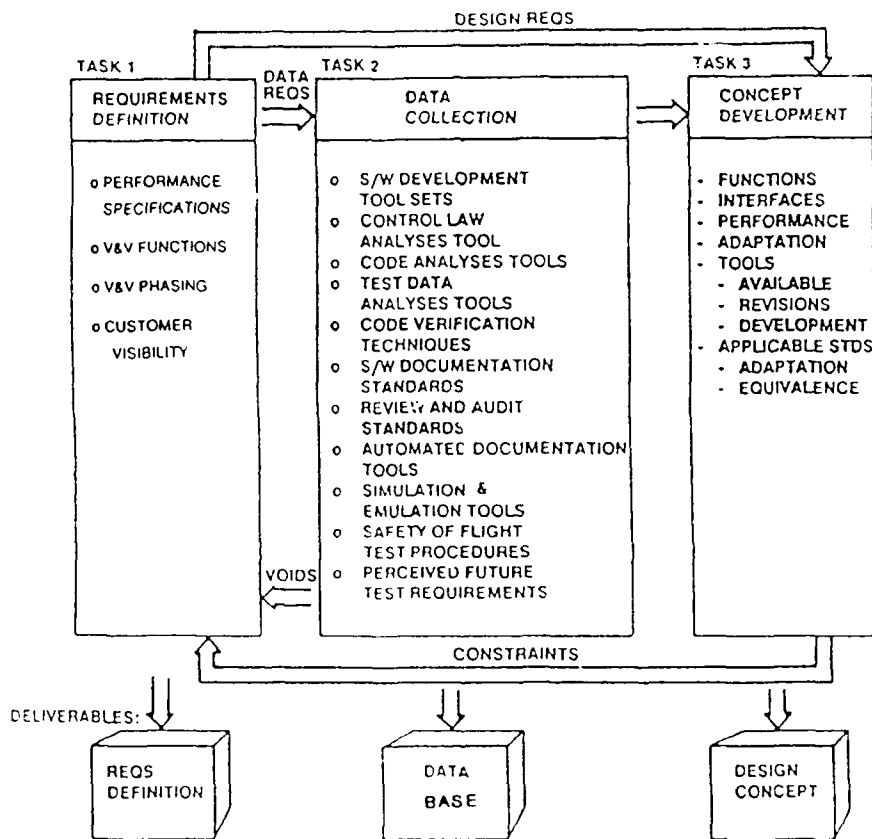


Figure 2.2-1 Overview of Program Approach

**2.2.1 Task 1 - Requirements Analysis.** The purpose of the requirements analysis task was to define a complete set of requirements which must be met in the V&V of flight critical systems software. The complexity of flight critical systems development has outpaced the management and technical resources supporting their acquisition. The rising complexity in hardware technology,

software technology, and the integration of systems stresses the capability to design, build, and test such systems. An initial draft of the requirements definition was prepared primarily from Frontier's experience in previous software V&V efforts. This draft provided a guide for areas of investigation during the Data Collection task. The information gained from the data collection task was then in turn used to update the requirements definition resulting in a complete and consistent set of requirements.

2.2.2 Task 2 - Data Collection. The approach taken in the data collection task was to first develop checklists (see Appendix) to help organize the acquisition of information on each element/function required in the software verification and validation methodology. The checklists addressed specifics about available software analysis tools and techniques available and current perceptions/experiences in the development of FCS software. Included within these checklists was a list of applicable Government standards which must be met in the development of flight critical systems. Examples of such standards include MIL-STD-483,-490, MIL-STD-2167A, MIL-STD-1521, ANSI/MIL-STD-1815A-Ada, MIL-F-9490, MIL-STD-8457, to name a few. One of the problems besetting software developers today is being knowledgeable in the requirements spelled out in many standards and even more importantly, being able to meet the intent of the standards in a cost effective manner.

Next, a catalog of applicable software tools used in all phases of development and verification and validation of flight critical systems software was prepared. This included control law analysis tools, software development tool sets, documentation aids and tools, configuration management software tools, and verification and validation software tools. As a starting point this list was compiled from our knowledge and hands-on experience with many of the tools. This list was then expanded through the information gained in literature searches, surveys of software tools used by flight critical system developers, and focused interviews of key government and industry experts. This list of tools was entered into our Tools Data Base Management System for ease of reference and quick recall.

The next step in the data collection process was to conduct in depth interviews with key government and industry experts in the development of flight critical systems. These interviews included personnel at Wright Research and Development Center -- Flight Dynamics Laboratory, Air Force Flight Test Center, NASA Dryden Flight Research Facility, McAir, General Dynamics, Rockwell International, Honeywell, Softech, and High Plains.

The prepared checklist was used to guide the discussions on FCS software verification and validation tools and techniques. At the completion of these interviews, the raw data findings were prepared and provided to WRDC/FIGX. As part of the data collection task, a Data Base Management System (DBMS) was developed to help organize and record the many sources of data and the tools and techniques identified. This DBMS was created on DBASE III+ and hosted on an IBM compatible PC. New data is continuing to be added to this system and it will serve as a reference source for proposed Phase II efforts.

2.2.3 Task 3 - Verification and Validation Concept Design. The third task in our technical approach to development of a V&V methodology was to synthesize a preliminary design in sufficient detail to demonstrate that the design is technically sound and that V&V functions have been defined to address the FCS software V&V requirements. A variety of tools and techniques were considered. The approach synthesized provides a user-friendly host environment for current and future V&V tools applications.

The scope of tools and techniques will allow the user to assess development of critical flight systems software from systems requirements definition on through "iron-bird" validation testing leading to flight test. Features which are included in the design are:

- o Verification and validation activities performed in a series of steps which are interfaced through the customer and coordinated with the development phases of the flight critical systems software.
- o Utilization of control systems analysis tools to verify flight critical systems software in a manner which will address safety of flight issues and provide a high level of confidence in the expected performance of the system design and implementation.
- o Use of both system level and statement level emulations for verification of flight critical systems software design. System level emulation will be used for analysis of performance, stability, and redundancy management. Statement level emulation will be used to exercise the code itself.
- o Structuring a verification and validation approach to provide an appropriate balance between external V&V activities and developer V&V efforts.
- o The use of proven Independent Verification and Validation (IV&V) techniques for evaluation of the prime contractor's development process and software design.
- o Use of advanced hierarchical modeling techniques for the development of topological network trees which are more easily understood than code representations, and which constitute a deliverable data base to serve as an effective tool for software change impact analysis.
- o Analysis techniques which will provide an assessment of overall system performance as well as verification of system/software design and coding.
- o Use of a structured data base of proven validation test procedures which have been used to perform "iron-bird" validation testing of highly integrated flight control systems. This data base addresses tests of performance, redundancy management, mechanization, failure mode and effects, aircraft systems and interfaces, and all other matters effecting flight performance and safety of flight.
- o Utilization of quick-look analysis tools applicable to ground based simulation and flight test data and application of advanced data reduction techniques.

A FCS software V&V methodology has been designed which provides these features and is discussed in Section 3.5. Specific techniques and tools are presented to address each of the phases involved in an V&V of flight critical systems software.

## 2.3 SUMMARY OF THE RESULTS

The results of this conceptual design study have shown that there is a growing need to develop a methodology by which flight critical systems software can be verified and validated for performance and safety impacts. The nature of the evolving technology and its application to FCS software verification and validation requirements has current V&V methods lagging behind design methods and tools. It is recognized throughout government and industry that FCS software V&V requires knowledgeable and skilled individuals utilizing proper tools and techniques to successfully complete the V&V effort in a timely manner. This report provides an overview of the development process of flight critical systems and the roles of verification and validation which go hand-in-hand with the development process. It provides a conceptual design for a computer aided environment to perform FCS software verification and validation.

### 2.3.1 Results of FCS Software V&V Requirements Definition and Data Collection

The process of defining V&V requirements is iterative and was performed in conjunction with our data collection effort. Initial draft requirements were prepared and updated as new information was gained from our interviews with key government and industry experts and through data gained in literature reviews. Key to the definition of these requirements was having a sound understanding of current FCS software development and V&V practices. Figure 2.3-1 is a representation of the development process with specifications of verification and validation levels.

A variety of accepted techniques are currently used to perform the V&V of FCS software. These techniques are supplemented with tools (manual procedures



or software programs) to aid in the analysis tasks and the bookkeeping of the results. Since the V&V of FCS software is normally performed as a parallel effort with the development efforts, accepted techniques used in the V&V efforts are driven by the timing availability of FCS software development products. The current V&V tasks, applied techniques, and V&V objectives are summarized in Table 2.3-1.

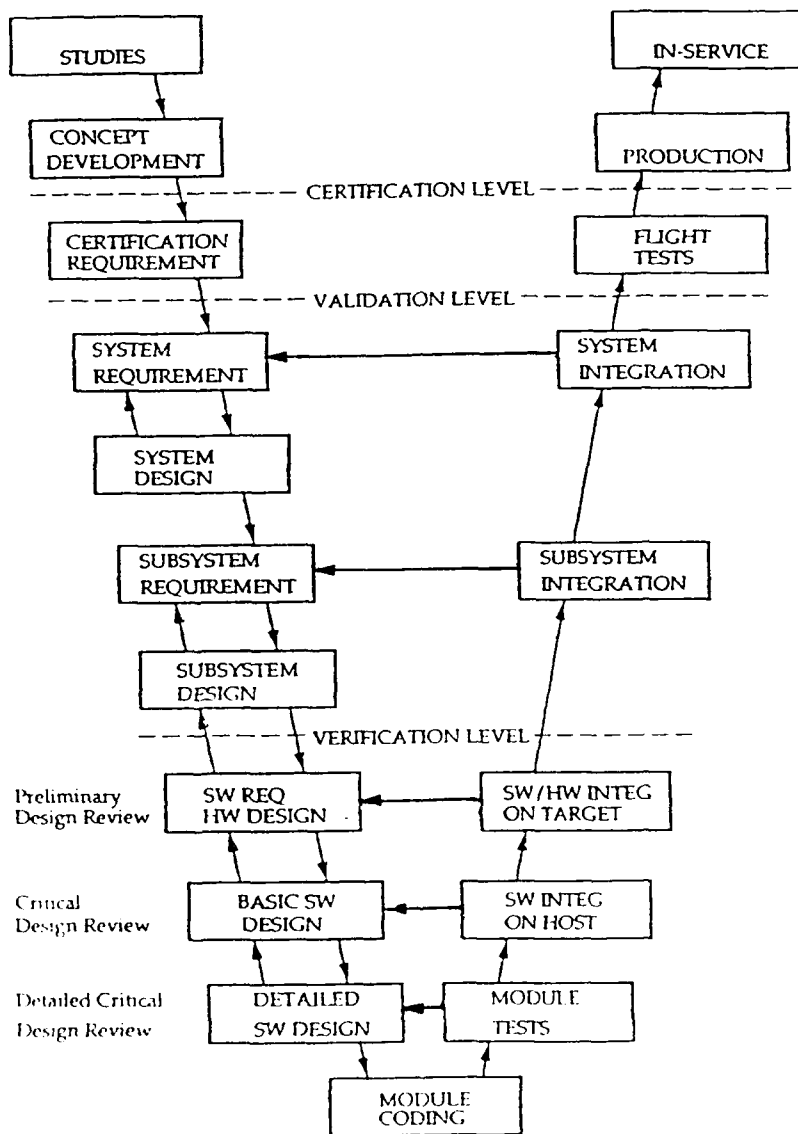


Figure 2.3-1 FCS Development and V&V Phases

# TABLE 2.3-1 V & V REQUIREMENTS

V&V TASKS	OBJECTIVE/PURPOSE	APPLICABLE TECHNIQUES/TOOLS
System Specification Verifications	Evaluated to ensure that system/subsystem considered will fulfill mission goals and objectives.	<ul style="list-style-type: none"> <li>o Requirements Analysis</li> <li>o Documentation Review</li> </ul>
Control Law Analysis	Assure control algorithms adequacy. Verify equation accuracy; evaluate functional relationships and functional performance (timing, sequencing, etc.)	<ul style="list-style-type: none"> <li>o Requirements Analysis</li> <li>o Control Law Analysis</li> <li>o Emulation/Simulation</li> </ul>
Evaluation of Development Planning	Evaluate for satisfactory standards & practices, schedules, planning, controls, reviews, audits, CM change control, problem resolution, V&V	<ul style="list-style-type: none"> <li>o Review Management Plan</li> <li>o CDDP Review</li> </ul>
Evaluation of Software Development Methodology	Preventive Measure. Sound design, coding, and test techniques reduce number of errors made during development.	<ul style="list-style-type: none"> <li>o Document Review <ul style="list-style-type: none"> <li>- Standards</li> <li>- Plans</li> <li>- Configuration Management Provisions</li> </ul> </li> </ul>
Software Requirements Verification	Requirements evaluated for adequacy, completeness, accuracy, testability, and traceability to higher level specifications.	<ul style="list-style-type: none"> <li>o Requirements Analysis</li> <li>o Critical Requirements Identification</li> <li>o Documentation Review</li> </ul>
Software Design	Evaluate development products to ensure technical viability and contribute to refinement process. Ensure software design represents a clear, consistent and accurate translation of software requirements.	<ul style="list-style-type: none"> <li>o Design Analysis</li> <li>o Performance Analysis</li> <li>o Document Review</li> <li>o Top Down Programming</li> <li>o System Level Emulation</li> <li>o Consistency Checker</li> <li>o Standardization</li> </ul>
Code Correctness	<p>Test and evaluate developers code using independent tools. Code is checked for errors, omissions and incorrect translations. Evaluate logic, file structuring, execution paths and limitations, interfaces, etc. Machine level open-loop tests and unit and module; closed-loop at subsystem/system. Examine timing.</p> <p>Identify unexpected paths for information flow through a program by analyzing the clues characteristic of sneak paths in network trees/flow graphs.</p>	<ul style="list-style-type: none"> <li>o Code Analysis</li> <li>o Comparator</li> <li>o Compiler</li> <li>o Interface Checker</li> <li>o Document Review</li> <li>o Cross Reference</li> <li>o Cross Assembler</li> <li>o Simulation</li> <li>o Instruction Trace</li> <li>o Sneak Analysis</li> </ul>
<p>Software Validation</p> <ul style="list-style-type: none"> <li>o Development Tests</li> <li>o System Tests</li> <li>o Flight Tests</li> <li>o Control Lane Response</li> <li>o Handling Quantities</li> <li>o Functional Tests</li> </ul>	<p>Determine whether all software and system performance, interface, functional and test requirements are fulfilled.</p> <ul style="list-style-type: none"> <li>- Every requirement is adequately tested</li> <li>- All subsystems are properly integrated</li> <li>- All system responses are adequate for performance and safety.</li> </ul>	<ul style="list-style-type: none"> <li>o Test Plan/Procedure Review</li> <li>o Test Case Generation</li> <li>o Hot-Bench Simulator</li> <li>o Mainframe Simulation</li> <li>o Iron-Bird Simulation</li> <li>o Aircraft Flight</li> </ul>
<ul style="list-style-type: none"> <li>o Redundancy Management</li> <li>o Failure Management</li> </ul>	<p>Insure, through independent testing, that RM/EM system meets design requirements for worst case combinations of failure; perform parametric analysis of pathological paths.</p>	<ul style="list-style-type: none"> <li>o System Level Emulation</li> <li>o Iron Bird Simulation</li> <li>o FNET</li> </ul>
Tool Development and Maintenance	Prepare a software tool set to aid in the performance of V&V assessment tasks.	<ul style="list-style-type: none"> <li>o Tool Evaluation</li> <li>o Tool Development</li> <li>o Tool Test &amp; Maintenance</li> <li>o Tool Documentation</li> </ul>

As can be seen, a large number of techniques have been developed to aid in the verification and validation of software. A deliverable Data Base Management System (DBMS) was implemented by FTI to help organize and record the many sources of data and the tools and techniques identified in this effort.

Evolution of Computer Aided Software Engineering (CASE) tools continues towards providing software development with the environment of an integrated tool set including planning, analysis, design, documentation, static analysis, prototyping, dynamic analysis, simulation, and construction of executable systems. Unfortunately, CASE tools have not reached the point where this broad application of tasks has been integrated into one development environment. Similarly, the use of CASE tools in a V&V environment offers great promise and some CASE tools have been specifically designed to perform reverse engineering on existing designs, a necessary V&V capability.

The near term trends in flight control systems (FLCS) are expanding the use of real-time, on-board optimization and intelligent controls to achieve high performance and provide for damage tolerance and self-healing designs. These near term FLCS already are addressing the inner-loop, outer-loop, and redundancy management functions. FCS integration has an even more challenging impact on software. Forecasts and projections for FCS in the 21st century indicate a number of significant impacts on FCS software V&V as indicated below.

- o Significant increases in computer power will cause major expansion in scope and character of onboard systems.
- o Development of architectural branches within redundant systems will add verification and validation complexity, embedded replicated or dissimilar subchannels for self monitoring could reduce redundancy management complexities at higher rates.
- o Increased throughput and emerging new architectures are allowing sensor fusion with information integration and display, requiring expanded FCS verification and validation roles.

- o Trends towards systems highly integrated through FLCS -- because of mission and performance benefits -- leads to more testing at system levels, interdisciplinary expertise, and pilot involvement.
- o Increase of control effectors and reduction in actuator redundancy levels for self repair/reconfigurable flight control increases the complexity of validation testing.
- o Decision-Aiding systems in a real-time environment require validation of knowledge bases which currently have no accepted validation methods.

The use of higher order languages (HOLs) eases the task of FCS design process. HOLs allow the flight control engineer to more easily follow the design through implementation in software. Current problems with Ada (tasking and rendezvous) do not stop its use; the specific problem areas can be avoided. However, there is still the question that once code is recompiled, it is difficult to say that new code is good versus assembly language patches approach.

FCS designers are moving towards providing control law block diagrams from which code can be generated automatically. GE's program called FASTER directly generates 1750A assembly code. Many of the linear analysis tools (Ctrl-C, Matrix X, MATLAB, etc.) purport to generate code directly from block diagrams which can then be used directly in simulations to test from design through simulation. Tools of these types eliminate many of the coding errors which can occur during the process of changing a flight critical design to code.

Future FCS systems will still have to address interfacing with existing, older systems. There is a wide generation of computers currently fielded and this will always be the case. Many of the older computers cannot support HOL. Development contractors are moving towards using RISC computers. However, currently there is not adequate support tools in this environment.

Transportable software is also being addressed. However, software compilers are currently a problem here because there is no agreed to standard.

Also, timing is one of the most critical elements in flight critical software and this effects transportability. Common module approaches requires that the developer look at real needs in V&V. Resources are currently being spent in designing software test stations/tools that will test to requirements and using language translators to implement new front-ends to these test stations/tools. Control law filters have already been transported. Use of Ada will help transportability in future developments.

Vehicle Management Systems (VMS) is the new focus in flight critical systems. However, the designers must be realistic about what they propose and use. It is cost prohibitive and not even possible to test all combinations required to "adequately" validate a highly integrated FCS.

Redundancy & Monitoring (R&M) is another design area which drives V&V requirements. From a design point, coverage of all failure mechanisms is the problem here. There is the question of quad vs triplex systems. Triplex systems can meet the  $1 \times 10^{-7}$  requirement, but it is difficult to meet an imposed requirement of fail-op, fail-op without going to a quad system. From a cost and development viewpoint, a quad voter runs twice as long as a triplex voter; software complexities at least double for every channel added due to combinatorial considerations.

A number of lessons learned can be gained from past efforts in development of flight critical systems software.

- o In general, problems arise in specifications across on-board aircraft systems. Understanding and documentation of interfaces between systems are often lacking.
- o The use of simulation for testing integrated systems is questionable. This is particularly true in the area of sensors. The FCS testing is dependent on models for high technology sensors and in this area modeling is very difficult. Systems integration adds more combinations of conditions which need to be tested.
- o People who have tested systems have to put information gained back into the loop. The problems that were encountered and how they were solved is often not reported.

- o One very large requirement is requirements and specifications for control laws. There is a lack of a reasonable MIL-Spec for flight control; PIO prediction is an example of this. Mil Prime Standard 8785-C is not adequate; it is a back-up guide.
- o Most errors are in design. These are generally found in systems integration testing. B-2 put a lot of time and money to get set up for systems integration testing and that has paid off well.

### 2.3.2 Conceptual Design for a V&V Methodology for FCS Software

The FTI technical approach to the development of the flight critical systems verification and validation methodology is based on a balanced allocation of technical skills, proven V&V tools and techniques, and evolving software developmental test methodologies. The implementation of our methodology can provide a workstation environment providing the needed tools and techniques for verification and validation of flight critical systems.

Proven V&V tools which can be used directly to meet the V&V requirements will be reviewed as candidates to be used in the development of a Computer Aided Verification And Validation Engineering System (CAV<sup>2</sup>ES). The CAV<sup>2</sup>ES will be hosted in a workstation environment and will provide the user with ready access to those requirements, design, and development details needed to assess the state of development of FCS software. Much of the early development verification activities will utilize V&V tools hosted in the workstation environment itself and will provide analysis data and results which can be carried from one V&V stage to the next. The V&V methodology will also address the validation test activities which take place when the FCS software development progresses to the point where testing of software and subsystems utilizes hotbench simulators, simulators requiring mainframes for computational support and finally when testing moves into an ironbird test environment.

The CAV<sup>2</sup>ES will provide an environment in which the flight control engineer or software engineer can quickly and easily access and analyze design information, software code, or generate data to verify and validate FCS software. It will allow the engineers to deal with all phases of the development

cycle and tackle the problem of maintaining a continuity of requirements/design evaluations across development phases.

The Computer Aided Verification and Validation Engineering System (see Figure 2.3.2-1) will provide the following functions: V&V Executive (VEXEC), Tool Interface Manager (TIM), User Interface (UI), Simulation Computer Interface (SCI), Data Recording System (DRS), Data Base Manager (DBM), Automatic Pilot Functions (APF), and the Data Analysis System (DAS).

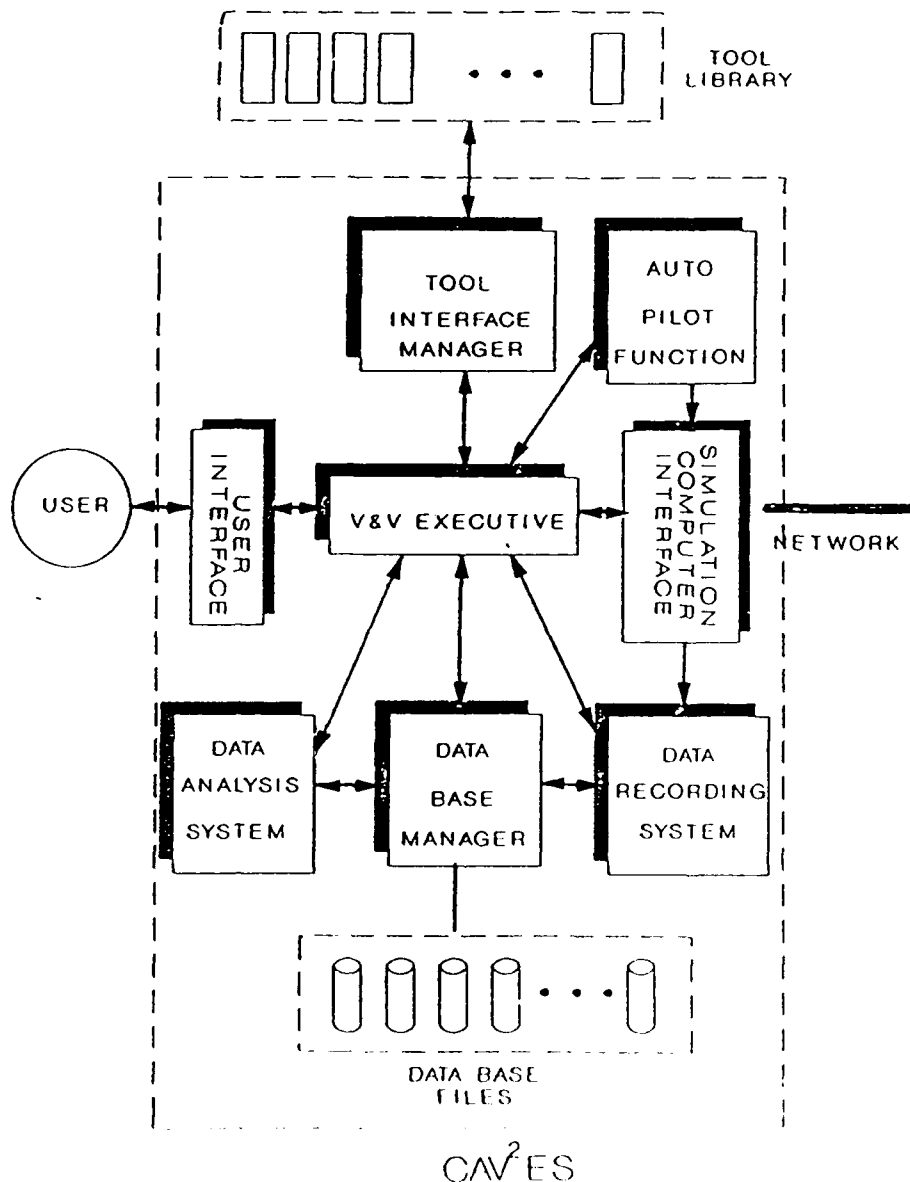


Figure 2.3.2-1 Computer Aided Verification and Validation Engineering System (CAV<sup>2</sup>ES)

To aid the FCS software analyst in performing V&V tasks, CAV<sup>2</sup>ES retains its own data base and library of tools. The data base is structured to be able to store and retrieve the data items which are a product of executing the verification and validation analysis tools. The tool library consists of two parts, a generic tools set and an external tools set. The generic tool set is a group of V&V tools which will perform basic V&V functions on FCS software. The external tools set represents user selected or new tools which need to be interfaced through the Tool Interface Manager.

The Tool Interface Manager (TIM) selects which interfaces are used with the selected tools so that the output of the library tool is put into a standard format acceptable to the Data Base Manager. If a new tool (external tool) is to be interfaced to CAV<sup>2</sup>ES, the TIM has an interface build capability which aids the user in building a functional interface which converts data to a format consistent with the existing data base.

The V&V Executive (VEXEC) monitors and coordinates the operation of CAV<sup>2</sup>ES functions. The VEXEC execution involves issuing commands to the Tool Interface Manager, the Simulation Computer Interface, the Data Recording System, the Data Base Manager, the Automatic Pilot Functions and the Data Analysis System. It receives commands and sends responses to the user via the User Interface.

The purpose of the User Interface (UI) is to provide direct access to the various software tools functionalities, while relieving the user from needing to be intimately knowledgeable about the software tools as stand-alone systems and adapting to their various styles and syntax. This means that a user who wants to obtain a time history plot of data generated by a simulation tool, CTRL-C or MATRIX-X for example, does not need to know the particular commands for the tool package for simulation and plotting. However, the UI does not confine the experienced tool user to stay within the UI interface, but provides a direct tool mode in which the user can execute tool commands within the CAV<sup>2</sup>ES environment to perform any simulation and plotting activity allowed by the tool. The UI is built in a windows environment to provide quick expansion or contraction of backup information, aiding in the verification and validation



process. The functionality of a tool can be accessed via point-and-click mouse operations on icons, menu, and form driven screens.

The Simulation Computer Interface (SCI) is used to communicate to the simulation computers. Communication may take place over serial lines to various devices, over ethernet, and over direct bus links. The user may open a terminal window for each of these connections and manually type commands. All commands, along with responses, will be logged and sent to the DBM to be recorded in a test execution log. Other subsystems of CAV<sup>2</sup>ES may also send commands to the simulation computers. All commands indicate if a response is expected. SCI will then pass along the command and wait for the response, if necessary.

The Data Recording System (DRS) is responsible for recording simulation data (both real-time and non-real-time) and transferring data to the data base manager. The DRS receives its commands from the VEXEC. Before a test begins, the VEXEC sends a list of commands to be executed (recording script). The real-time simulation recording takes place via a link (bus link or ethernet) connected to the simulation computers via SCI. Proper synchronization is critical if a valid set of data is to be recorded.

The Data Base Manager (DBM) serves two purposes. First, to create and maintain data base files and second, to conduct data transactions for other CAV<sup>2</sup>ES subsystems. The DBM is composed of two processes to serve these purposes: Vexec Interface and Build Update. The interface process conducts transactions while Build Update is used to create and maintain data base files. To assist in performing these processes, a commercially available data base management system, UNIFY, will be used.

The Automatic Pilot Functions (APF) subsystem provides the capability for the CAV<sup>2</sup>ES system to send pilot commands to the aircraft flight control system. The APF provides the capability to provide flight test functions (FTF) for testing of performance parameters and to fly fundamental maneuvers. VEXEC obtains commanded maneuvers from the test procedures via the Data Base Manager. The APF sends these commands to the simulation computers and flight control system via the SCI. The APF uses a transportable auto-pilot model

which may be hosted on the simulation computers (mainframes) or in the CAV<sup>2</sup>ES workstation environment, dependent on the particular type of communications link used between the CAV<sup>2</sup>ES and the simulation computers. This provides flexibility in the choice of this communication link from one implementation to the next.

The Data Analysis System (DAS) is designed to provide the validation engineer the capability to examine the data recorded during a test and to perform data reduction techniques on the recorded data. The recorded data can be displayed both on a CRT and on a hardcopy device. Data displayed during simulation execution will generally be plots of variables as a function of time and mode switches. Post test data analysis can be performed providing performance parameters of flight critical systems.

### 2.3.3 CAV<sup>2</sup>ES FCS V&V Capabilities

Capabilities to be included within the CAV<sup>2</sup>ES environment will provide the flight control/software engineers the capability to assess the FCS software design in terms of performance, stability, and redundancy management analysis. It will provide both static and dynamic code analysis tools for verification and validation of flight critical systems code. It will also provide the capability to perform quick-look analysis of generated data. It will provide the means to verify and validate FCS software through control of real-time simulators driven by proven test procedures/test cases.

Aircraft Flight Critical Systems Analysis. The CAV<sup>2</sup>ES will provide the capability to evaluate the adequacy of the control laws with respect to performance and stability and to evaluate system mechanization with respect to redundancy management, timing and bus loading. To perform these analysis, 3 general types of flight control analysis tools will be used: a linear analysis and design tool, a nonlinear simulation tool, and a system level emulator. Commercially available linear analysis and design tools provide a comprehensive interactive control design and analysis software language system including state-of-the-art primitives in classical and modern control synthesis, matrix

analysis, dynamic system analysis, parameter estimation, and graphical presentation.

A general purpose non-linear aircraft simulation program incorporates specific aircraft characteristic through user-defined modules for the aerodynamic forces and moments, the propulsion system, the control system, etc. It can be run to trim the aircraft for any desired flight condition, to generate linear state models for the trimmed flight condition, and to generate time history responses for user-defined inputs representing commands or external disturbances.

A system level emulator can be used to analyze the correctness of module logic and functions, bus loading, timing and redundancy management, as well as analyzing the operational capabilities of a system and its conformance to system requirements.

Software Design Verification. The objective of software design is to confirm the technical adequacy of the design. Much of this effort involves manual review of requirements and design specifications, requiring methodical and diligent attention in matching requirements to design and in evaluating designs. It is in this area where the application of CASE tools can provide benefits in verification. In general, CASE tools leverage the requirements analysis and design specification phases of the software development cycle while more traditional tools are more applicable in the software implementation phase.

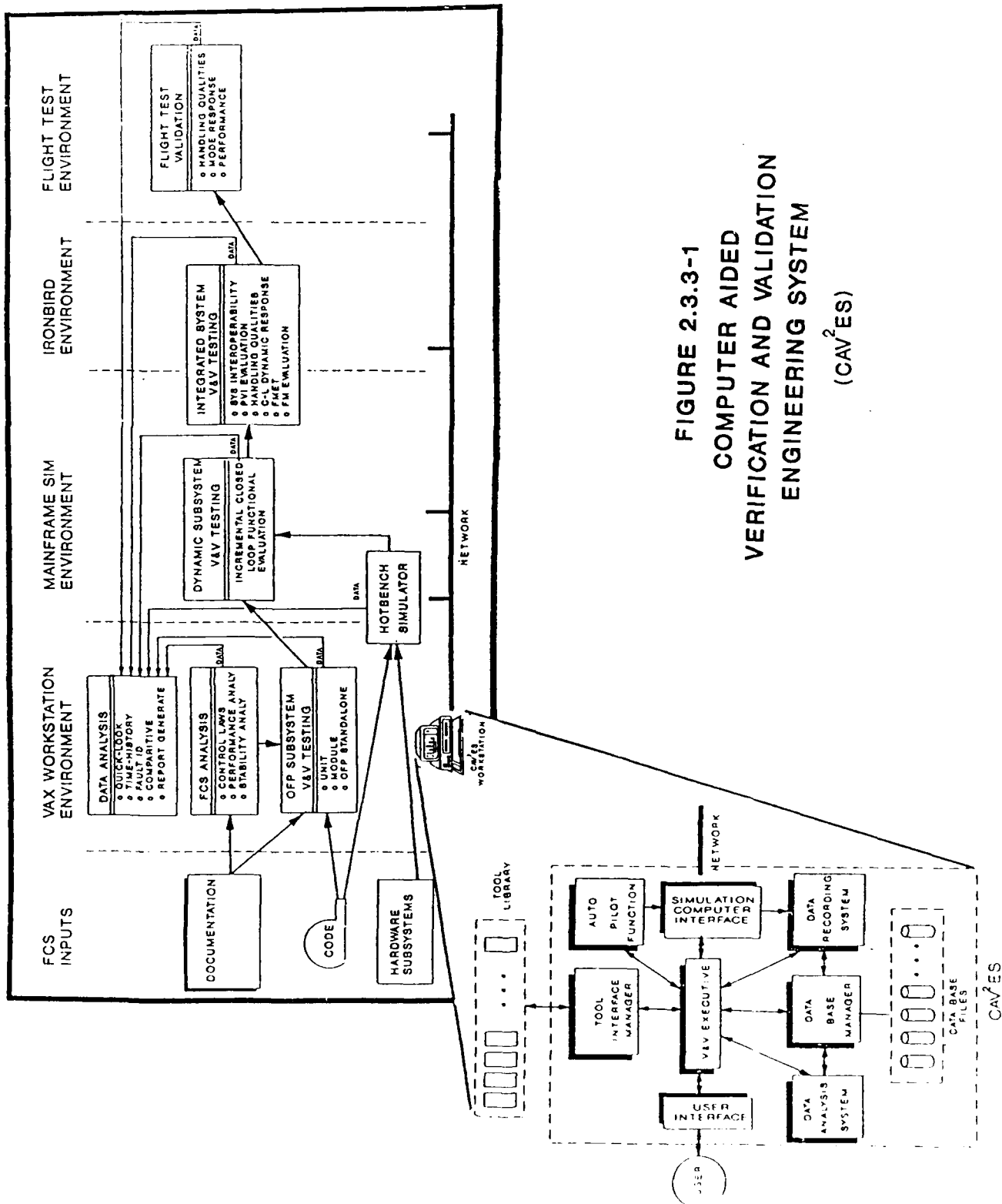
FCS Software Code Verification. Specific static analysis techniques which will be initially implemented will be selected as a part of the Phase II effort. However, capabilities to perform software sneak analysis and to perform instruction level emulator code execution are examples of static and dynamic analysis tools, respectively, which provide a strong code verification tool base and are planned for implementation in CAV<sup>2</sup>ES.

Stand Alone, Dynamic Subsystem, and Integrated System V&V. The engineering and formal system and software testing utilizes a bottom-up philosophy.

Testing starts with a lowest level code (unit code) and progresses upward to system-level testing. Verification of results is performed at each level before progressing to the next level. Test plans and procedures are prepared for each level of formal testing and test results are documented. Problems or discrepancies detected during any phase of testing are documented, investigated, and acted upon.

CAV<sup>2</sup>ES will be used to communicate to simulation computers and subsystems to drive individual subsystem tests, integrated subsystem tests, and finally system level tests. The Maneuvering-simulation Validation Automation (MVA) system currently under development by FTI for SAAB-SCANIA will provide just this type of capability. The design structure of CAV<sup>2</sup>ES will incorporate many of the features in the MVA system.

The CAV<sup>2</sup>ES is an integrated V&V tool set to be used throughout FCS software development phases. It's integration and application with the FCS development phases is shown in Figure 2.3.3-1. It is usable by developers, V&V organizations, and research groups. Its workstation provides the user with a powerful computational environment which can utilize many of the more sophisticated FCS design and analysis tools. CAV<sup>2</sup>ES provides a CASE type working environment, user friendly interfaces, and provides hooks for addition of future tools. It provides a means for interfacing and driving real-time simulations and offers an environment aiding automation of real-time testing. In short, the CAV<sup>2</sup>ES is an evolutionary system which accommodates new tools to meet the growing user requirements in verification and validation of flight critical systems software.



## 2.4 POTENTIAL APPLICATIONS OF THE CAV<sup>2</sup>ES DEVELOPMENT EFFORT

The Computer Aided Verification and Validation Engineering System has many commercial applications which will bring large savings to its users. A great deal of interest has been expressed by government agencies and industry leaders in providing new capabilities in the verification and validation of the evolving flight critical systems. The CAV<sup>2</sup>ES will be directly usable by industry developers of systems which are flight critical and are driven by software. It will also be usable by manufacturers of automated systems controlled by software which can effect the performance of their product and the safety of their user.

The aircraft manufacturers including General Dynamics, McAir, Rockwell, Lockheed, Northrop, and Grumman all currently are faced with the problems of developing and integrating flight critical systems as their major product output. They have expended a large amount of resources in attempting to build and maintain verification and validation capabilities to keep pace with the increasing growth in the use of software in virtually all of their applications. The CAV<sup>2</sup>ES offers these industry leaders an inexpensive tool which can be easily implemented and customized to meet their specific needs. Of course, the commercial market extends beyond the United States market and covers our NATO allies and supported neutral countries such as Sweden. Companies such British Aerospace (BAE), Messerschmitt-Boelkow-Blohm (MBB), SAAB-SCANIA, and GEC Avionics are all heavily involved in the same pursuit of developing, verifying, and validating flight critical systems.

CAV<sup>2</sup>ES will be of use to the many subcontractors to the prime aircraft manufacturers. These include: the developers of flight control systems such as Honeywell, General Electric, and Lear Siegler; engine manufactures such Pratt & Whitney, General Electric, and Rolls Royce; and other manufactures of subsystems such as hydraulic actuators, braking systems, and the large number of integrated avionics components going into modern day aircraft. All of these developers of subsystems face the problem of verification and validation of software used to control and interface their subsystems with other flight

critical systems. This requires the verification activities involved in the early phases of design and development of software, and also requires that they provide and a realistic environment with which to test their subsystems, validating that they meet specified performance goals and safety requirements.

The interest and application of a CAV<sup>2</sup>ES in the government is readily apparent by simply reviewing the many procurement and research activities being announced on a daily basis. Examples of these include solicitation by: NASA Ames Research Center (SBIR: 03.10 - "Development, Testing and Verification Of Flight Critical Systems"); DARPA (SBIR: 90-087 - "Low Cost Reconfigurable Generic Computer Workstations for Simulation Research/Development/Analysis"); Air Force (SBIR: A90-470 - Verification and Validation of Expert Systems); and a recent procurement announced by the Avionics Laboratory titled "Advanced Avionics Verification And Validation". The CAV<sup>2</sup>ES has capabilities which are directly applicable to each of these and offers the flexibility to expand to future required capabilities. The direct use of the CAV<sup>2</sup>ES by the many research groups in the government organizations is not limited to WRDC but extends to all government services engaged in the research and development and testing of flight critical systems software.

CAV<sup>2</sup>ES offers a capability to the independent software development and software verification and validation contractors who the government solicits to perform IV&V activities on flight critical systems. For contractors such as these, the CAV<sup>2</sup>ES offers a low cost test bed tool which can be applied at all phases of verification and validation. CAV<sup>2</sup>ES is a tool which the SPOs can offer or specify as a support tool with the knowledge that it is a tool which can produce the type of products that are needed to validate FCSs safety requirements.

Aside from the contractors who build flight critical systems, the CAV<sup>2</sup>ES offers capabilities of interest to commercial manufactures of systems which use software that can effect the user's safety. The car manufactures have braking system, engine systems, and ride quality systems which are controlled by software and could impact safety if failure occurs. Developers of robotic

controlled manufacturing processes also need a capability to develop, verify, and validate software which is safe to use in a manufacturing environment.

In summary, the usefulness and numbers of applications of the CAV<sup>2</sup>ES is very large including: all branches of the military government agencies supporting the development of flight critical systems; the aircraft manufacturers and their many subcontractors; the companies who provide independent and consulting support in the verification and validation of flight critical systems; and other commercial manufacturers who use software as an automating and integrating means in the development of their commercial products. The resources saved on future FCS programs are many times the CAV<sup>2</sup>ES development costs.



## SECTION 3.0

### TECHNICAL DISCUSSIONS

#### 3.1 DEVELOPMENT PHASES FOR FLIGHT CRITICAL SYSTEMS

##### 3.1.1 Overview

The development process of flight critical systems has many variations in literature, but the basic states are universal. We choose to follow that defined in Reference 1. The stages are:

- o Study phase
- o Concept phase
- o System requirements phase
- o System design phase
- o Subsystem requirements phase
- o Subsystem design phase
- o SW requirements phase-----HW specification phase
- o Basic software design phase
- o Detailed software design phase
- o Coding/module test phase-----HW development phase
- o SW integration on host computer phase
- o SW integration on target computer phase-----HW integration phase
- o Subsystem integration phase
- o System integration phase
- o Flight test phase
- o Production phase
- o Postdevelopment support or in-service phase

Our main emphasis in this study starts at the system requirements phase and carries through system integration up to flight test. Before embarking on a discussion of the development process, it is helpful to see how the roles of verification and validation go hand-in-hand with the development process. Figure 3.1-1 is a representation of the development process with specifications of verification and validation levels. Starting at the bottom of the figure, verification may be defined as the demonstration that each step in the design/development process (left side of Figure 3.1-1) is correct and that the software program is a correct rendition of the software design. Validation, shown in the middle of the figure, may be defined as the demonstration of the

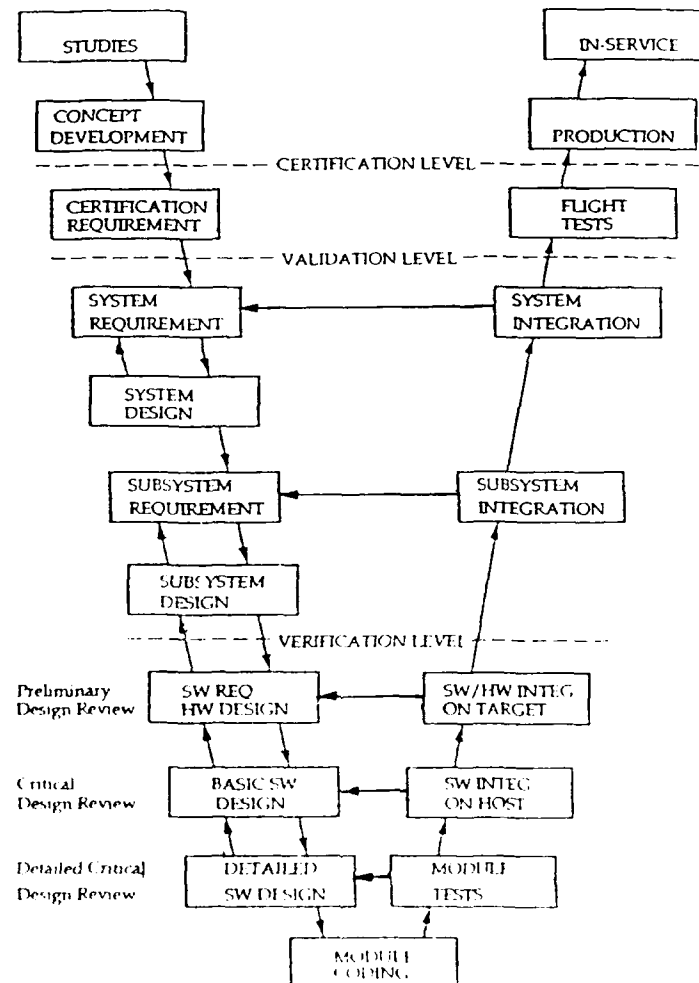


Figure 3.1-1 FCS DEVELOPMENT AND V&V PHASES

correct performance of the entire subsystem/system. With this brief introduction to V&V, we will return to a discussion of the presented development phases.

### 3.1.2 Systems Requirements/Design Phase

The system requirements/design phase produces a System Specification containing customer and other known requirements which must be later validated during the system integration efforts. In addition to the System Specification, a System Mechanization/Architecture document is normally produced describing the entire system structure, connections between subsystems, preliminary data, and design of the system. These documents then form the bases for requirements and design of subsystems.

### 3.1.3 Subsystem Requirements/Design Phase

During the subsystem requirements/design phase, each subsystem will normally have a Subsystem Requirements/Design Specification containing:

- o A functional description of the subsystem including a copy of the specific subsystem functional requirements contained in the System Specification
- o The subsystem design concept and subsystem architecture
- o Subsystem criticality classification
- o Requirements on:
  - Sensors and effectors
  - Processors
  - Displays and controls
- o A detailed description of the subsystem's interface with all other subsystems and with the system as a whole. Each subsystem will generate requirements which will be implemented in software or hardware. Detailed requirements for the hardware and software components to be used to implement the subsystems will be derived from the functional requirements during this phase. Some requirements for

the hardware components may be determined by use of certain software components and by the design environment.

- o The computing power (speed and memory) required to execute the software within the time allowed.
- o The availability of a particular software development environment. Also, certain functions might be required that can realized only through equipment specific software.
- o Built-In Test (BIT).
- o Data format conversation.
- o Specific sensor functions (e.g., Inertial Navigator, Display).

#### 3.1.4 Software Requirements/Hardware Specification Phase

Separation for subsystem functions between hardware and software components is carried out for each subsystem during the Software Requirements/Hardware Specification Phase. Precise boundaries between hardware and software cannot be determined in an effective way prior to completion of the Subsystem Requirements and Subsystem Design Phases. Results of the Software Requirements and Hardware Specification Phase are normally recorded in:

- o Software Requirements Documents (including descriptions and derivations of all algorithms to be implemented)
- o Equipment Specifications
- o An Interface Definition and Control Document

#### 3.1.5 Basic Software Design Phase

The Basic Software Design Phase transforms the software requirements for a subsystem into a software module design and a subsystem test design. Program structure, program flow, module functions, and interfaces with other modules are determined during this phase. Timing considerations are normally addressed within a framing concept defining the module calling sequences based on experience from existing similar systems. Bus load analysis should be confirmed by rapid prototyping tests.

Results of the preliminary software design process should be contained in a Software Design Specifications and a Software Test Requirements Document. These documents are verified against the appropriate Software Requirements Document during formal review at the end of the Software Preliminary Design Phase.

### 3.1.6 Detailed Software Design Phase

During the Detailed Software Design Phase, the internal module structure, the algorithms to be implemented, and the data structures to be used are completely defined. The internal module structure is described in functional flows or pseudo-code. Also, module test procedures are prepared.

The following information developed during the Detailed Software Design Phase is added to the Software Design Specification.

- o Module descriptions
- o Detailed functional flows
- o Detailed data flows
- o Detailed interface and data description
- o Module test specifications.

At the end of the Detailed Software Design Phase, a Critical Design Review is held to confirm that:

- o The design is complete. All internal modules are present. All interfaces are defined.
- o All software requirements have been satisfied.
- o The required software quality attributes are achieved.

### 3.1.7 Module Coding/Test Phase

The Software Design Document and Software Test Document form the basis for module coding and test activities, respectively. During the Module Coding/Test Phase, the modules are coded and commented; syntax errors are debugged, and

code inspection and code walk throughs are performed. All previously specified module tests are prepared. Test stubs and drivers are established. Static and dynamic module tests are executed. Detected errors are corrected.

The Software Design document is updated to be reflective of the implemented design and a listing of source code is added to the documentation. The Software Test Document procedures are completed to form a module test report.

### 3.1.8 Software and Software/Hardware Integration Phase

During this phase, the process is to integrate the individual modules into a working subsystem. This is normally accomplished on a host development computer. Inputs and outputs to each module are simulated and the modules are successively integrated one at a time into a software subsystem. When all modules are integrated, the software subsystem is moved to a target computer and the code is retested. Software/Hardware test procedures are filled in and bound to provide a Software/Hardware Integration Test Report.

### 3.1.9 Subsystem Integration Phase

At this point the subsystem code is moved to the target computer (flight computer). The software components that handle the hardware interfaces to the flight computer are integrated one at a time and tested. Finally, the Subsystem Test Procedures are executed on the flight computer and a test report of results is prepared.

### 3.1.10 System Integration Phase

The System Integration Phase is performed in a similar manner to the subsystem integration. Subsystems are integrated individually into the system and tested. The end task of the system integration is to validate that the overall system satisfies the system requirements in the System Specifications. The system tests performed at this phase are normally very extensive and cover all of the functional and performance capabilities to be achieved at the different points in the flight envelope. When the system has been adequately

validated for system safety properties, and handling qualities and performance, the system is moved into flight test.

### 3.1.11 Flight Critical System Engineering

One function which is particularly important in system testing, but extends back to the beginning of the system design is that of flight critical systems engineering. With the growth in integration of flight critical systems, there is a growing need to provide a flight critical systems engineering function to administer to the needs of the flight critical portion of the aircraft. The functions which fall into FCS engineering can be summarized as follows:

- o Allocate aircraft requirements to Vehicle Management System (VMS) flight critical requirements. This includes definition of architecture, redundancy levels, fault coverage requirements, functional partitioning, iteration rates, system time delays, information flow, through-put and memory requirements, avionics system interface requirements, and flight test aids requirements.
- o Administer to external interfaces of flight critical systems. This includes providing a support function in terms of interface definition, expertise/consulting on interfaces during development of FCS and providing a configuration control function.
- o Provide development services in terms of requirements interpretation during design, a "fireman's" role during detailed design, and in general provide coherent progressive processes for system development.
- o Provide expertise in system level testing by assuring that the testing level is established at the level the requirements are generated supporting system or subsystem level and provide a system level test leadership role.

The role of the flight critical systems engineering function then is to provide expertise and control functions in the requirements, design, implementation, and test of all FCS on the aircraft.

### 3.1.12 Flight Test Phase

The task of the Flight Test Phase is to carry out all testing required to certify safety and performance characteristics in the actual flight environment.

## 3.2 VERIFICATION AND VALIDATION OF FCS SOFTWARE

### 3.2.1 Overview

The purpose of verification and validation is to provide systematic assurance that FCS computer programs will perform their mission requirements efficiently and correctly. The V&V effort serves as a program acceptance tool in providing higher confidence in software reliability, compliance between specifications and code, and adherence to accepted standards. When performed by an independent party, independent V&V (IV&V) provides the customer better visibility into the development effort, a second source of technical expertise, better document quality and reduced frequency of operational change.

Proper performance of V&V on FCS software requires an understanding of software V&V tools and techniques. It also requires a strong theoretical background, and experience in FCS design and analysis. This background and experience are needed for use of proper analysis techniques and putting enough emphasis on critical functions. There are unique problems associated with the design and development of digital flight critical systems. For example, in some cases, direct discrete design is required as opposed to discretization of an analog design. However, it is also necessary to account for pure time delays introduced by the digital design. In a Digital Flight Control System (DFLCS), the validity of the design should be verified throughout the operating envelope. Simple frequency response measurements are not generally adequate for this purpose, particularly where variable gain scheduling is utilized (and if task-tailored control laws or reconfigurable control laws are to be implemented).

The V&V process is designed to address each critical phase of the software development process. Software development is comprised of many subactivities or



tasks and the V&V process assures that each development task has been completely and correctly performed. A comprehensive V&V effort to assure software reliability will include the following basic tasks:

1. Requirements analysis: assure that software requirements have been correctly derived from system requirements and that the hardware/software interface requirements are compatible.
2. Design analysis: assure that the proposed design is feasible and that proposed mathematical equations and algorithms will satisfy the software requirements.
3. Code analysis: assure that developed code is a correct implementation of the software design.
4. Testing: assure proper operation of program modules, software interfaces, and system performance.

A detailed description of each V&V activity is presented in the following paragraphs.

### 3.2.2 FCS Software Requirements Analysis

The definition of software requirements is one of the most critical phases of the software development process. Verification of software requirements is performed to ensure that system and interface requirements (documented in the system and subsystem specifications) are correctly allocated to software requirements (documented in the Computer Program Development Specification). The criteria employed in this evaluation include completeness, correctness, and testability.

Several techniques have been successfully applied to the verification of software requirements. These techniques include:

- o Independent derivation of software requirements from system/subsystem requirements.

- o Comparison to standard reference systems or similar systems previously developed.
- o Functional simulations and modeling of process allocation.
- o Timing and sizing analysis, and the establishment of budgets for flight critical system parameters.
- o Development of a requirements chart which identifies interrelationships between requirements.

A V&V test criteria is selected for use in confirming proper implementation for each valid software requirement. The results of this analysis are presented in a Software Requirements Analysis Report and any problems detected are documented and acted upon.

### 3.2.3 FCS Software Design Analysis

After system and subsystem requirements have been allocated down to software, the software design phase can begin. This is the process of translating software requirements into a basic software design and then a detailed software design. It is imperative to verify that the proposed software design satisfies all the software requirements.

The software design defines both the executive control logic and algorithms to perform each software function. A balance of analysis techniques must be selected to verify both of these elements of the software design. The following design analysis techniques have proven effective in detecting design errors:

- o Correlation and traceability between design elements and software requirements.
- o Functional simulation to assess design integrity and process allocation.
- o Independent derivation of equations and algorithms.
- o Comparison with standard references and models.
- o Comparison with methods which have been proven in operational systems.
- o Mathematical and logical analysis.

Design analysis techniques to be utilized for any particular function are dependent upon the nature of the function (such as signal filtering, gain scheduling, device interfacing). For example, logic analysis techniques are appropriate for executive control functions while mathematical methods are more suited for numerical functions. The proposed design of each software function is verified by using the selected method to determine the extent to which it satisfies the corresponding software requirements. Control logic is similarly verified to ensure proper interaction between software functions.

#### 3.2.4 Code Analysis

Analysis of the developed program code is performed to ensure that the coded representation of the software design corresponds to the verified design. The goals of the program analysis are to ensure that the coding is correct, that development standards have been followed, and that no latent errors have been introduced into the software by the coding process. The following program analysis techniques are examples of those employed to identify coding errors:

- o Version comparisons
- o Text editing and syntax analysis
- o Standards auditing
- o Equation reconstruction
- o Data structure analysis
- o Flowcharting and logic reconstruction
- o Manual code inspection
- o Software sneak analysis

Software tools, which are programs designed to assist the analyst, are employed to automate many of the above program analysis techniques. Software tools can be used to help identify actual or potential errors in the developed code, and reformat and consolidate information. They present a reliable, cost-effective means to greatly reduce the manual program analysis techniques.

To maximize the visibility of the software development, program analysis is performed in parallel with the code development. This is achieved by analyzing the incremental code deliveries and modifications introduced in the updated program versions. This method has proven to be an effective technique for identifying major coding problems and for correction early in the code development process. Any discrepancies between the final code and the verified software design are documented and acted upon.

#### 3.2.5 Flight Critical Systems Software Test

To complete the validation, tests are performed to determine compliance with software and system requirements. A comprehensive test plan is developed prior to testing and tests are planned to achieve the following objectives:

1. Verify that individual software functions satisfy the corresponding software requirements.
2. Verify that the software/software and hardware/software interface functions are properly implemented.
3. Verify that the operational system possess the required system capabilities and satisfies the appropriate performance requirements.

Tests are planned at module, interface, and system levels for both nominal and extreme conditions within the required performance limits. Test procedures are developed to provide a detailed specification of the exact steps to be used in performing the test. The results of the test planning activity are presented in a Test Plan/Procedures document. Comprehensive planning is the foundation upon which effective testing is based.

Testing is conducted by following the exact procedures specified in the Test Plan/Procedures. Software tools are employed during the tests to provide a testing environment, an acceptance criteria, and analysis aid. Test results are

recorded, and any anomalous results are confirmed by analysis, documented and acted upon.

Testing of FCS software generally occurs at five major levels during software integration and test phases of a system development. These are:

1. Execution on the host computer.  
(Module Testing)
2. Execution on an emulator of the target computer.  
(Module & Interface Testing)
3. Execution on the target computer.  
(Module & Interface Testing)
4. Integrated system simulation testing.  
(Interface and System Testing)
5. Flight testing.  
(System Testing)

These five levels are normally used in most FCS software development; the last three levels are always required. Execution on a host computer usually refers to a development type environment, where code can be examined in a static environment, modules can be tested, and structure of the code can be tested for proper interfaces between modules.

When the target computer is not available, the target computer is emulated on a host computer. This emulation testing uses identical instruction sets, word sizes, etc. yet provides a host computer user friendly environment in which test drivers and analyzers can be "wrapped around" the emulator.

Execution on the target computer provides the actual computer environment for the software execution. Individual modules are tested and integrated to a complete subsystem/system to allow end-to-end testing. Execution on a target computer in conjunction with some external inputs (real and modeled hardware devices) is often referred to as "hot-bench" testing. One of the primary functions of "hot-bench" testing is to check external interfaces and software functional performance to realistic real-time inputs.

Integrated system simulation testing is performed to check the FCS software in the full range of operational conditions. In this phase of testing, prototype or actual flight computers are brought into a hot-bench, iron-bird, and/or test rig environment. The environment is made to present the operational environment as close as is practical. This environment includes high-fidelity aerodynamics, sensors (normally simulated), hydraulics and actuators, cockpit, instrumentation, and outside view presentations for pilot-cueing. The closed-loop tests that are run are generally pilot-in-the-loop operations to verify performance, flying qualities, and to confirm proper functional dynamics and mode sequencing. In addition to these tests, "pilot confidence" testing is performed where the pilots fly realistic missions and push the simulated aircraft to its safety limits.

Flight testing is directed toward confirmation of performance requirements and demonstrating flight safety. Considerable instrumentation is needed to collect data which can be analyzed and correlated with analytical and simulator predictions.

### 3.2.6 FCS Software V&V Tools and Techniques

Table 3.2.6-1 broadly summarizes FCS software V&V tasks, applicable V&V tools and techniques, and the V&V objectives. An extremely large number of tools have been and continue to be developed to aid in the verification and validation of software. A variety of these tools (static and dynamic) are listed in Table 3.2.6-2. Static tools examine some aspect of specifications, designs, or code without executing the code. These tools are grouped into a list of those which examine a specific property and those which examine more general and extensive properties. A dynamic tool performs some function to aid in testing the software when the program is actually executed. A timing analyzer that monitors and records execution times for functions is an example of a dynamic tool.

Techniques are "standards and procedures" used in development, test, and maintenance of software. Table 3.2.6-3 presents some standard techniques used. Development and maintenance techniques are included since substantial software reliability can be obtained by attention to systematic development and documentation.

# TABLE 3.2.6-1 V & V REQUIREMENTS

V&V TASKS	OBJECTIVE/PURPOSE	APPLICABLE TECHNIQUES/TOOLS
System Specification Verifications	Evaluated to ensure that system/subsystem considered will fulfill mission goals and objectives.	<ul style="list-style-type: none"> <li>Requirements Analysis</li> <li>Documentation Review</li> </ul>
Control Law Analysis	Assure control algorithms adequacy. Verify equation accuracy; evaluate functional relationships and functional performance (timing, sequencing, etc.)	<ul style="list-style-type: none"> <li>Requirements Analysis</li> <li>Control Law Analysis</li> <li>Emulation/Simulation</li> </ul>
Evaluation of Development Planning	Evaluate for satisfactory standards & practices, schedules, planning, controls, reviews, audits, CM change control, problem resolution, V&V	<ul style="list-style-type: none"> <li>Review Management Plan</li> <li>CPDP Review</li> </ul>
Evaluation of Software Development Methodology	Preventive Measure. Sound design, coding, and test techniques reduce number of errors made during development.	<ul style="list-style-type: none"> <li>Document Review <ul style="list-style-type: none"> <li>Standards</li> <li>Plans</li> <li>Configuration Management Provisions</li> </ul> </li> </ul>
Software Requirements Verification	Requirements evaluated for adequacy, completeness, accuracy, testability, and traceability to higher level specifications.	<ul style="list-style-type: none"> <li>Requirements Analysis</li> <li>Critical Requirements Identification</li> <li>Documentation Review</li> </ul>
Software Design	Evaluate development products to ensure technical viability and contribute to refinement process. Ensure software design represents a clear, consistent and accurate translation of software requirements.	<ul style="list-style-type: none"> <li>Design Analysis</li> <li>Performance Analysis</li> <li>Document Review</li> <li>Top Down Programming</li> <li>System Level Emulation</li> <li>Consistency Checker</li> <li>Standardization</li> </ul>
Code Correctness	<p>Test and evaluate developers code using independent tools. Code is checked for errors, omissions and incorrect translations. Evaluate logic, file structuring, execution paths and limitations, interfaces, etc. Machine level open-loop tests and unit and module; closed-loop at subsystem/system. Examine timing.</p> <p>Identify unexpected paths for information flow through a program by analyzing the clues characteristic of sneak paths in network trees/flow graphs.</p>	<ul style="list-style-type: none"> <li>Code Analysis</li> <li>Comparator</li> <li>Compiler</li> <li>Interface Checker</li> <li>Document Review</li> <li>Cross Reference</li> <li>Cross Assembler</li> <li>Simulation</li> <li>Instruction Trace</li> <li>Sneak Analysis</li> </ul>
<p>Software Validation</p> <ul style="list-style-type: none"> <li>Development Tests</li> <li>System Tests</li> <li>Flight Tests</li> <li>Control Lane Response</li> <li>Handling Quantities</li> <li>Functional Tests</li> </ul>	<p>Determine whether all software and system performance, interface, functional and test requirements are fulfilled.</p> <ul style="list-style-type: none"> <li>Every requirement is adequately tested</li> <li>All subsystems are properly integrated</li> <li>All system responses are adequate for performance and safety.</li> </ul>	<ul style="list-style-type: none"> <li>Test Plan/Procedure Review</li> <li>Test Case Generation</li> <li>Hot-Bench Simulator</li> <li>Mainframe Simulation</li> <li>Iron-Bird Simulation</li> <li>Aircraft Flight</li> </ul>
<ul style="list-style-type: none"> <li>Redundancy Management</li> <li>Failure Management</li> </ul>	<p>Insure, through independent testing, that RW/EM system meets design requirements for worst case combinations of failure; perform parametric analysis of pathological paths.</p>	<ul style="list-style-type: none"> <li>System Level Emulation</li> <li>Iron Bird Simulation</li> <li>FMT</li> </ul>
Tool Development and Maintenance	Prepare a software tool set to aid in the performance of V&V assessment tasks.	<ul style="list-style-type: none"> <li>Tool Evaluation</li> <li>Tool Development</li> <li>Tool Use/Documentation</li> <li>Tool Maintenance</li> </ul>

TABLE 3.2.6-2 VERIFICATION AND VALIDATION TOOLS

<u>SPECIFIC STATIC TOOLS</u>	<u>GENERAL STATIC TOOLS</u>	<u>DYNAMIC TOOLS</u>
CIRCULAR REFERENCE CHECKER	ACCURACY ANALYZER	DATA FLOW PATHING
CODE COMPARATOR	DOCUMENTATION AND CONSTRUCTION SYSTEMS	EMULATION
CROSS-REFERENCE CHECKER		SIMULATIONS
DATA BASE ANALYZER	EDITOR	• COMPUTER
FLOW CHARTER	FORMAL LANGUAGES WITH SYNTAX ANALYZERS	• HYBRID
INTERFACE CHECKER		• TEST BED (IRON B-RD)
MODULE INVOCATION	• REQUIREMENTS	TEST DATA GENERATOR
PROGRAM FLOW ANALYZER	• SPECIFICATIONS	TEST DRIVER
SET/USE CHECKER	• PROGRAM DESIGN	TEST EXECUTION MONITOR
UNITS CONSISTENCY CHECKER	• PROGRAM CODE	TEST RECORD GENERATOR
UNREACHABLE CODE VECTOR	SNEAK-PATH ANALYZER	TIMING ANALYZER
	SYMBOLIC EVALUATOR	
	THEOREM PROVER	

#### TOOL DEFINITIONS

Accuracy analyzer - analyzes numerical calculations for req'd accuracy  
Circular reference checker - modules calling each other  
Code comparator - differencing between versions  
Cross-reference checker - calling of modules; external variables called  
Data bases analyzer - module accesses to data bases; unused elements  
Data Flow Pathing - trace execution sequence for variable(s) in flow  
Documentation & constructions - Auto documentation; Consistent data pool  
Editor - Analyze/extract information/relationships from source programs  
Emulations - System level model generated from requirements, not design  
Flow charter - show logical construction of program  
Formal Languages - Program structures and rules  
Interface Checker - Check Range, limits, scaling of variables  
Module Invocation Tree - Establishes call hierarchy with system  
Program flow analyzer - statistics on usage; estimate execution time  
Set/Use checker - Checks for variables: set, not used; & used before set  
Simulations - Test characteristics, algorithms, functions, performance  
Sneak-Path Analyzer - Looks for unexpected paths  
Symbolic evaluator - reconstructs equations relating output to input  
Test data generator - produces test cases to exercise the system  
Test driver - controls the execution of a program  
Test execution monitor - collects data and compares to expected results  
Test record generator - analyzes, reduces, and formats results  
Theorem prover - axioms used prove assertions stated for a path  
Timing analyzer - monitors/records run time of functions and routines  
Units consistency checker - variable expressions (units) checked  
Unreachable code detector - looks for code which cannot be executed



TABLE 3.2.6-3  
DEVELOPMENT AND V&V TECHNIQUES

Abstractions and hierarchies to reduce complexity: abstractions such as trees are used to make the design simple and clearly defined  
Checkout (debug) testing: function/module testing before integration  
Constructive design approaches: (eg. formal design language)  
Critical Design Review: oral demonstration of detailed design  
Data flow diagram, structure chart: shows flow of data in program and hierarchical organization  
Descriptions or documentation  
Design guidelines, test guidelines, & coding guidelines  
Design standards, coding standards  
Functional capabilities list: module description of functions to perform  
Integration testing: code test after modules are assembled; I/O structure  
Organization as finite automata: provides clear structure of functions for FLCS  
Qualification audit  
Singularities and extremes testing  
Symbolic execution: performed on special functions such as mode logic  
Systems concept review: oral demo of initial concepts, trade-offs, etc.  
Validation testing: final demo in simulation environment

Evolution of Computer Aided Software Engineering (CASE) tools continues towards providing software development with the environment of an integrated tool set which includes planning, analysis, design, documentation, static analysis, prototyping, dynamic analysis, simulation, and construction of executable systems. (See Reference 2) Table 3.2.6-4 presents some of the applicable CASE tools that are commercially available. These tools span the gamut from powerful linear systems analysis, prototyping and code generation to those which provide aids in the form of a data dictionary, creating data-flow diagrams, process specifications, and graphic documentation of design. Evolving CASE tools are providing ways to help manage the complexity of large-scale software systems. The tools, like the methods they implement, are not final solutions but are aids to providing a more friendly software development and test environment.

Table 3.2.6-4 REPRESENTATIVE CASE TOOLS, METHODOLOGIES, AND LIFE CYCLES

CASE TOOL	METHODOLOGY	LIFE CYCLE
TEAMWORK OS/2.3.0	DeMARCO, WARD/ SCHLAIR, CONSTANTINE	PLANNING, ANALYSIS DESIGN
EXELERATOR 1.84	YCURDON, GENE/SARSON	PLANNING, ANALYSIS DESIGN
POSE 4.0	YOURDON, GANE/SARSON, CONSTANTINE, FINKELSTEIN, INFORMATION ENGINEERING	PLANNING ANALYSIS DESIGN, CONSTRUCTION
TRACEBUILDER II	TRACES SOFTWARE REQ'S FORWARD & BACKGROUND	ALL PHASES OF SOFTWARE DEVELOPMENT
INTEGRATED SYSTEMS, INC	LINEAR MODELING, SYSTEM BUILD, AUTO-CODE GENERATION	CONCEPTUAL DESIGN, LINEAR ANALYSIS PROTOTYPING

Besides attempting to ease the software programmer's burden, code reusability is another issue which CASE developers are targeting. Software now constitutes 90% of an electronic systems functionality (vs. 10% during the 1960s). With so much code being written, CASE tool manufacturers are developing tools to tackle reusability problems. These tools are part of a larger system of front-end tools to streamline the programming process and make it more efficient.

Fundamentally, CASE tools must meet several criteria in order to be successfully adapted as part of a software developer's tool kit. These criteria include:

- o Break down complexity of requirements and designs into manageable components.
- o Presentable to several audiences including end-users and contracting organizations.
- o Cheaper than building the real thing as compared to conventional software development approach.
- o Quantitative and Verifiable with respect to requirements traceability and performance criteria.
- o Graphically oriented to provide more easily understood graphical illustrations of design.

### 3.3 TASK 1 RESULTS: REQUIREMENTS FOR FLIGHT CRITICAL SYSTEMS SOFTWARE V&V

#### 3.3.1 General Requirements for Flight Control Systems

General requirements for flight critical systems are generally broken into two parts: mission requirements and safety requirements. For example, flight control/engine control is normally first driven by safety requirements and secondly by mission requirements. For most other systems, mission requirements are given more design attention relying on flight control (and the pilot) to provide the needed margins of safety. Commonly used specifications for probability of loss of an aircraft due to a flight control system failure is:

$$P_{\text{loss}} < 1 \times 10^{-7} \text{ per flight hour (military aircraft)}$$

$$P_{\text{loss}} < 1 \times 10^{-9} \text{ per flight hour (civil aircraft).}$$

With the advent of digital flight control systems, these failure probabilities have been applied to the total digital system including the software. To achieve these numbers for the entire flight control system safety, the probability of aircraft loss due to software failure must be even lower to achieve this high safety margin. Emphasis in past and current developments has been placed on fault avoidance and fault tolerance techniques in software design.

With the advent of the highly coupled flight critical systems, this distinction between flight control and other flight critical systems regarding safety consideration is diminishing and fault avoidance/tolerance issues must be applied in all flight critical systems.

#### 3.3.2 FCS Design Trends Impact On Software V&V Requirements

Design techniques which are currently used to address safety aspects of FCS software include fault avoidance and fault tolerance. Fault Avoidance techniques apply structured design methods that incorporate rigorous

quality control and systematic testing of the software to insure that the probability of a software "bug" being introduced or remaining undetected during the software design and development process is extremely low. One technique of fault avoidance is the use of very small and very simple modules which are relatively easy to verify. It is commonly assumed that high integrity can be achieved through use of such techniques. In practice, however, no matter how carefully the software is designed, it is impossible to establish that it is completely error free because: (1) the larger number of possible states preclude exhaustive testing; and (2) the usual statistical analysis methods which are useful in hardware development are not applicable to software development. Therefore, Fault Tolerance is introduced into design to cope with faults which are not discovered during design and implementation process. A good fault tolerant design should prevent any remaining faults from having a catastrophic effect on the system.

Flight control systems requirements have and will continue to drive safety aspects in software design. While requirements for digital flight control are not unique, collectively, they represent the most demanding requirements in guidance and control applications. Real-time closed-loop operations, multi-mode design, bandwidth variations, multi-loop design, and tight interface/reliance on sensor systems are some of the characteristics of flight controls which drive overall software design complexity. These factors along with the trends in aircraft systems and avionics systems design toward the increased use of relaxed static stability and integrated avionics/control functions, place even more importance on software fault avoidance and fault tolerance.

The near term trends in flight control systems (FLCS) are expanding the use of real-time, on-board optimization and intelligent controls to achieve high performance and provide for damage tolerance and self-healing designs. These near term FLCS already are addressing the inner-loop, outer-loop, and redundancy management functions shown in Table 3.3-1.

TABLE 3.3-1 NEAR TERM TRENDS IN FLCS FUNCTIONS

FUNCTIONS		
INNER-LOOP	OUTER-LOOP	REDUNDANCY MANAGEMENT
<ul style="list-style-type: none"> <li>• RELAXED STATIC STABILITY</li> <li>• GUST LOAD ALLEVIATION</li> <li>• RIDE QUALITY</li> <li>• FLUTTER MODE CONTROL</li> <li>• INTEGRATED CONTROL</li> </ul>	<sup>2</sup> <ul style="list-style-type: none"> <li>• TF/TA/OA</li> <li>• AI BASED DECISION MAKING</li> <li>• INTEGRATED CONTROL</li> <li>• OPTIMAL FLIGHT PATH CONTROL</li> </ul>	<ul style="list-style-type: none"> <li>• ANALYTIC REDUNDANCY</li> <li>• AI BASED TECHNIQUES</li> </ul>

FCS Integration has an even more challenging impact on software. Table 3.3-2 summarizes some of the areas related to integration and the related impacts of fault tolerance.

Table 3.3-2 EXAMPLES OF IMPACTS OF INTEGRATION CONCEPTS ON COMPLEXITY

CONCEPT	DESCRIPTION	CRUCIAL FUNCTION IMPACT
DISPERSED, INTEGRATED FLIGHT/NAVIGATION SENSORS	SHARING OF STRAPDOWN NAVIGATION SENSORS WITH FLIGHT CONTROL	MORE COMPLEX FCS ALGORITHMS FOR SENSOR NORMALIZATION, REDUNDANCY MANAGEMENT, & SURVIVABLE DISPERSION
INTEGRATED FLIGHT/PROPULSION CONTROL	SHARING OF INFORMATION BETWEEN ENGINE AND FLIGHT CONTROL SYSTEMS, VECTORED THRUST	FLT CRUCIAL ENGINE CTRL
FLIGHT PATH MANAGEMENT	REAL TIME OPTIMAL CONTROL FOR TF/TA, COMPLEX ROUTE DECISIONS AT LOW ALTITUDE	LOW ALTITUDE AUTO FLIGHT MANAGEMENT IS CRUCIAL, SENSOR BLEND CONCEPTS-FLT CRUCIAL, HUGE TERRAIN DATA BASES (eg DTM) ARE FLIGHT CRUCIAL
VEHICLE MANAGEMENT SYSTEMS	<ul style="list-style-type: none"> <li>◦ IF P C</li> <li>◦ UTILITIES SYSTEMS MGMT</li> <li>◦ INTEG CONTROL FUNCTIONS</li> <li>◦ INTEG MAINT/DIAGNOSTICS</li> </ul>	<ul style="list-style-type: none"> <li>◦ COMPLEXITY</li> <li>◦ HIGHLY INTERACTIVE</li> <li>◦ RECONFIGURATION</li> </ul>

### 3.3.3 Technology Impacts on V&V Requirements

Technology advancements in flight critical systems software design and in software verification and validation testing have been significant over the past 15-20 years. The development and use of digital systems in flight critical systems applications have pushed verification and validation techniques to meet the demands of testing increasingly complicated systems. A number of accepted validation testing methods are used, but verification and validation technology generally lags the advances being made in the development of FCS software. Currently, convenient verification and validation methods and tools are lacking for multi-channel and highly integrated systems.

Trends and projections in flight control system design and impacts on validation have been presented in Reference 3 and are summarized in Table 3.3-3. The trends presented in the table are very realistic and provide motivation for developing improved FLCS verification/validation techniques simultaneously with evolving flight critical systems concepts. It has long been advocated that many of the complications associated with V&V of FCS software can be avoided by anticipating the V&V requirements early in the design process and by using many of the evolving structured V&V techniques and tools discussed in Section 3.5.

### 3.4 TASK 2 RESULTS: DATA COLLECTION

The primary efforts involved in the data collection task were to perform literature reviews on FCS software development and V&V methods and to conduct numerous focused interviews with FCS developers and key government FCS experts. In order to help organize and focus these efforts, FTI developed checklists (see Appendix) addressing the classes of information to be gathered. The checklists addressed specifics on current developmental approaches used for FCS software, available software analysis tools and techniques being used, facilities and support requirements, problems most often encountered during development, experiences in the development of FCS software, and perceptions of what FCS software development programs would require because

---

Table 3.3-3 Forecasts/Projections for FCS in 21st Century Indicate  
(source AGARD WG09)

- Significant increases in computer power will cause major expansion in scope and character of onboard systems
  - Development of architectural branches within redundant systems will add verification and validation complexity
  - Redundancy in management functions (e.g., voting planes, etc.) embedded in special purpose HW isolated from FLCS will change verification and validation complexity
  - Highly fault-tolerant HW designs that provide "dynamic redundancy" changes the scope and complexity of verification and validation efforts
  - Embedded replicated or dissimilar subchannels for self monitoring could reduce redundancy management complexities at higher rates
  - Increased throughput and emerging new architectures are allowing sensor fusion with information integration and display, requiring expanded FCS verification and validation roles
  - Trends are towards systems highly integrated through FLCS because of mission and performance benefits -- leads to more testing at system levels, interdisciplinary expertise, and pilot involvement
  - Increase of control effectors and reduction in actuator redundancy levels for self repair/reconfigurable flight control
  - High bandwidth FCS for active vibration and load control have associated characteristics which impact other FCS
  - Hypersonic vehicles require VMS to have total vehicle energy/thermal/trajectory management integrated with FLCS
  - Decision-Aiding systems in a real-time environment require validation of knowledge base which currently has no accepted validation methods
  - Interfaces and internetting to unmanned vehicles leads to additional complexities and verification and validation requirements
  - Boundaries between non flight critical and flight critical systems are projected to dissolve with increasing integration of systems
-



of the latest technology being incorporated into FCS. The checklists also included a list of applicable Government standards which address FCS software development. A catalog of applicable software tools used in development, verification and validation of flight critical systems software was prepared and included in the checklists to aid with tool identification.

Literature searches were performed including review of publications and recent articles at the AFTECH Library, review of software development and flight critical systems technical journals, and a Defense Technical Information Center search on related subjects. Hundreds of related article abstracts were scanned for appropriate subject material. In order to provide an orderly way of tracking and retaining pertinent information gained from these reviews a Data Base Management System (DBMS) was implemented. This DBMS was used then to help organize and record the many identified sources of data, tools and techniques. This DBMS was created on DBASE III+ and hosted on an IBM compatible PC.

A number of the reports and documents specifically directed at flight critical systems development and its verification and validation were identified. Two very recent AGARD reports, "Language Support Environments For Guidance And Control Systems" - Final Report Working Group 08, and "Validation Of Flight Critical Control Systems" - Report of GCP Working Group 09, were very helpful and current on many of the flight critical systems requirements, flight critical systems trends, development approaches, and verification & validation practices. The "Handbook - Volume I Validation of Digital Systems in Avionics and Flight Control Applications" and the "Digital Systems Validation Handbook Volume II", (References 4 and 5, respectively) both published by the US DOT are also excellent sources of material for verification and validation testing practices used in modern flight critical systems. Another informative reference that serves as a good primer on flight control software validation is the "Digital Flight Control Software Validation Study", (Reference 6) an AFFDL technical report. Discussions of recent Computer Aided Software Engineering tools were presented in several articles; the book by A.S.Fisher titled "CASE" gave an excellent summary of where CASE tools are now and their current trends.

In depth interviews were conducted with key government and industry experts in the development of flight critical systems. These interviews included personnel at Wright Research Development Center Flight Dynamics Laboratory, Air Force Flight Test Center, NASA Dryden Flight Research Facility, McAir, General Dynamics, Rockwell International, Honeywell, Softech, and High Plains. The prepared checklist was used to guide the discussions on FCS software verification and validation tools and techniques.

Comments gathered from the above sources have been summarized below in terms of V&V Drivers, Development and V&V Methodologies, Higher Order Languages, Development and V&V Tools, Validation Testing, Future FCS Software Considerations, and Problems/Lessons Learned.

#### FCS SOFTWARE V&V DRIVERS

- o FCS move to digital implementation increases complexity of V&V effort.
- o Increased systems integration accomplished through software.
- o Move to Ada requires update of V&V tools and techniques.
- o Complexities of Failure Management / Self Healing requires careful test planning to get adequate testing coverage.
- o Maintenance of FCS software requires extensive V&V capabilities.
- o Move towards transportability (MIL-STD-1750 kills this area).
- o Integration of software with hardware complicates V&V testing.

#### FCS SOFTWARE DEVELOPMENT

- o 2167A waterfall chart represents how development/test is performed.
- o Rapid prototyping is useful early design aid.
- o Projects organized along the lines of how the development proceeds are desirable. One good example is:.
  - Aerodynamic Stability and Control
  - Control Law Design and Analysis
  - Flight Critical Systems Engineering
  - Flight Control Mechanization and Software
  - Flight Control Hardware Design

- Flight Control Systems Test
- Flight Control Operations
- o Flight Critical Systems Engineering function is an increasingly important function to administer to needs of flight critical portion of aircraft. It provides a continuum of understanding across the development organization.
- o Quality Functional Deployment is good formal planning and documenting why you have done what you have done.
- o Some developers feel strongly that the software development should be kept with the flight control engineers. In principal, other developers agree, but that the software engineer is better equipped to write software -- therefore training software engineers in the development of FCS software is mandatory.

#### USE OF HIGHER ORDER LANGUAGES (HOLS)

- o HOLs are in general good. However, once code is recompiled, it is difficult to say that new code is good versus an assembly language patches approach.
- o One advantage of HOL is that it allows the system analyst (flight control engineer) to read or even develop the code. This avoids the problem of miscommunication between the designers and the software implementors.
- o The proper place to standardize is the language. Ada has some problems (Ada tasking, rendezvous, etc.), but you do not have to use all of the capabilities of the language.

#### FCS SOFTWARE DEVELOPMENT AND V&V TOOLS

- o FCS software development is moving towards provision of control law block diagrams to the FLCS houses for automatic code generation. GE's program called FASTER directly generates 1750A assembly code.
- o FLCS Tools used:
  - Ctrl-C
  - Matrix X
  - GenAir: Generic Aircraft, a McAir tool. This uses general

control laws and general aircraft configuration performance. Used for mission performance evaluation.

- Modular Design & Analysis Tools
- Nonlinear aircraft model simulation
- MATLAB, EASY5, MATRIXx, CTRL-C
- Use of script files: set of command files that go to simulation computers
- Recording system for all parameters in the simulation system.
- o Test tools that use actual flight boxes and automate testing are evolving. The Fully Automated Tester and Error Reporting (FATER) is an example which compares control laws in Fortran versus assembly.
- o TAE (Transport Application Executive) is a NASA Goddard Flight Center tool used in Simulated Rapid-Prototyping Facility (SRF) of the WRDC Flight Dynamics Laboratory. It has standard I/O for a program and runs on a dozen different computers.
- o FCS Integration Tools/Methods Include:
  - Basic documentation tools.
  - A lot of simulation for R&M testing.
  - In-house fault tree analysis.
  - 1553 analysis tools.
  - Flow diagrams & analyzing timing between functions.

#### FCS VALIDATION TESTING

- o Utilizes a bottom-up philosophy.
- o Starts with lowest level code and progresses to system-level testing.
- o Provides verification at one level before progressing to the next.
- o Test plans/procedures are prepared for each phase/level of testing.
- o Test results documented, discrepancies documented, investigated and acted upon.
- o Approach ensures system operates as designed and is flight worthy.

- o Provides total visibility of system development which allows better management control.
- o Handling qualities quantitative solutions have failed.
- o Integrated system V&V evaluates system functional requirements:
  - closed loop test environment simulates the dynamic behavior of the air vehicle
  - actual flight hardware is used where practical
  - verifies closed loop dynamic response
  - used for handling qualities evaluation
  - evaluate pilot vehicle interface evaluation
  - used for failure management evaluation
  - evaluates failure modes and effects tests
- o The ability to test back-to-back software (ie. previous OFP vs updated OFP) offers many benefits during validation.

#### FUTURE FCS SOFTWARE CONSIDERATIONS

- o Future FCS systems will most likely have to address interfacing with existing systems. A wide generation of FCS computers exists. Some older ones cannot support HOLs.
- o Development contractors are moving towards using.
  - RISC computers; currently there is not adequate support tools in this environment.
  - Ada language programming.
- o Transportable software is being addressed.
  - Software compilers are currently a problem here.
  - Timing is one of the most critical elements in flight critical software and this effects transportability.
- o Vehicle Management Systems (VMS) is the new focus in FCS
  - Developers must be realistic about what they propose and use.
  - The combinatorial considerations make it impossible to test all combinations
- o Developers are looking at real needs of common Module approach.
  - Designing test stations that will test.

- Using language translators for new front-ends to test tools.
- FLC filters have already been transported.
- Ada will help "Common Module's" in the future.
- o Redundancy & Monitoring (R&M)
  - Test coverage is the problem here.
  - There is the question of Quad vs Triplex. Triplex can meet the  $1 \times 10^{-7}$  problem, but it is difficult to meet a requirement of fail-op, fail-op without going to a Quad system -- is this requirement unduly imposed on flight critical systems?
  - A quad voter runs twice as long a triplex voter.
  - Software complexities at least double for every channel added.
- o Total System Integration
  - New techniques are being used for robust control laws, multi-axis, integrated flight propulsion methods, multi-thrust vectoring, and self repairing.
  - Need a manageable way of dealing with reconfiguration & fault isolation, reconfiguration control, and advance control design software.

#### PROBLEMS AND LESSONS LEARNED

- o Problems arise in specifications across flight critical systems interface.
- o Use of simulation for testing integrated systems is questionable.
  - Can not simulate EO and radar devices that well. Models can be built for it, but usually they are single thread.
  - Sensors & integration depend on models for high technology sensors. Modeling is very difficult.
- o Use of simulation for V&V
  - Organizations: people and equipment have to be planned and adequate.
  - There is always a reluctance to change a simulator once things are up and running. Some flexibility is required.

- o People who have tested systems have to put information back into the loop. The problems that were encountered and how they were solved is not reported. Only the good part/results seem to get published.
- o Result of flying qualities testing has produced much disinformation.
- o One very large need is requirements & specifications for control laws.
  - There is a lack of a reasonable MIL-Spec for flight control.
  - PIO prediction is an example of this.
  - Mil Prime Standard 8785-C is not adequate, it is a back-up guide.
- o Design group practices could be improved:
  - Must think ahead as to the way things will be tested.
  - Lack of documentation: integrated system documentation defining how systems work together is needed.
- o Most errors are in design. These are generally found in systems integration testing.
  - B-2 put a lot of time and money to get set up for systems integration testing and that has paid off well.
  - Verification of software is done very well. Few code errors now appear. Automating tests is easy.

### 3.5 TASK 3 RESULTS: DEVELOPMENT OF THE FCS V&V METHODOLOGY

#### 3.5.1 Technical Approach

The FTI technical approach to the development of the flight critical systems verification and validation methodology is based on a balanced allocation of technical skills, proven V&V tools and techniques, and evolving software developmental test methodologies. The implementation of our methodology will provide a workstation environment providing the needed tools and techniques for verification and validation of flight critical systems. This approach will address the growth in the use of software as the implementing and integrating media for the development of highly integrated flight critical systems. Our overall technical approach for developing and implementing the methodology is illustrated in Figure 3.5.1-1.

It is built to address the flight critical systems software development tasks and will provide timely evaluations for each development milestone. Our approach addresses each of the development phases and breaks out the verification and validation tasks, tools, and techniques which most appropriately can be used to evaluate the development efforts at that phase. A large data base of analysis, verification, and validation tools is available. Appropriate tools will be chosen to address the V&V requirements. Those tools which can be used directly to meet the requirements will be candidates to be used in the development of a Computer Aided Verification And Validation Engineering System (CAV<sup>2</sup>ES) which can be applied to the current development environment phase. For each tool evaluated that does not meet specific criteria, deficiencies which must be corrected will be identified and estimates of the effort required to correct these deficiencies will be made.

The CAV<sup>2</sup>ES will be hosted in a workstation environment and will provide the user with ready access to those requirements, design, and development details needed to assess the state of development of FCS software. Much of the early development verification activities will utilize tools hosted in a workstation environment and will provide analysis data and results which can be carried from one stage to the next. The V&V methodology will also address the



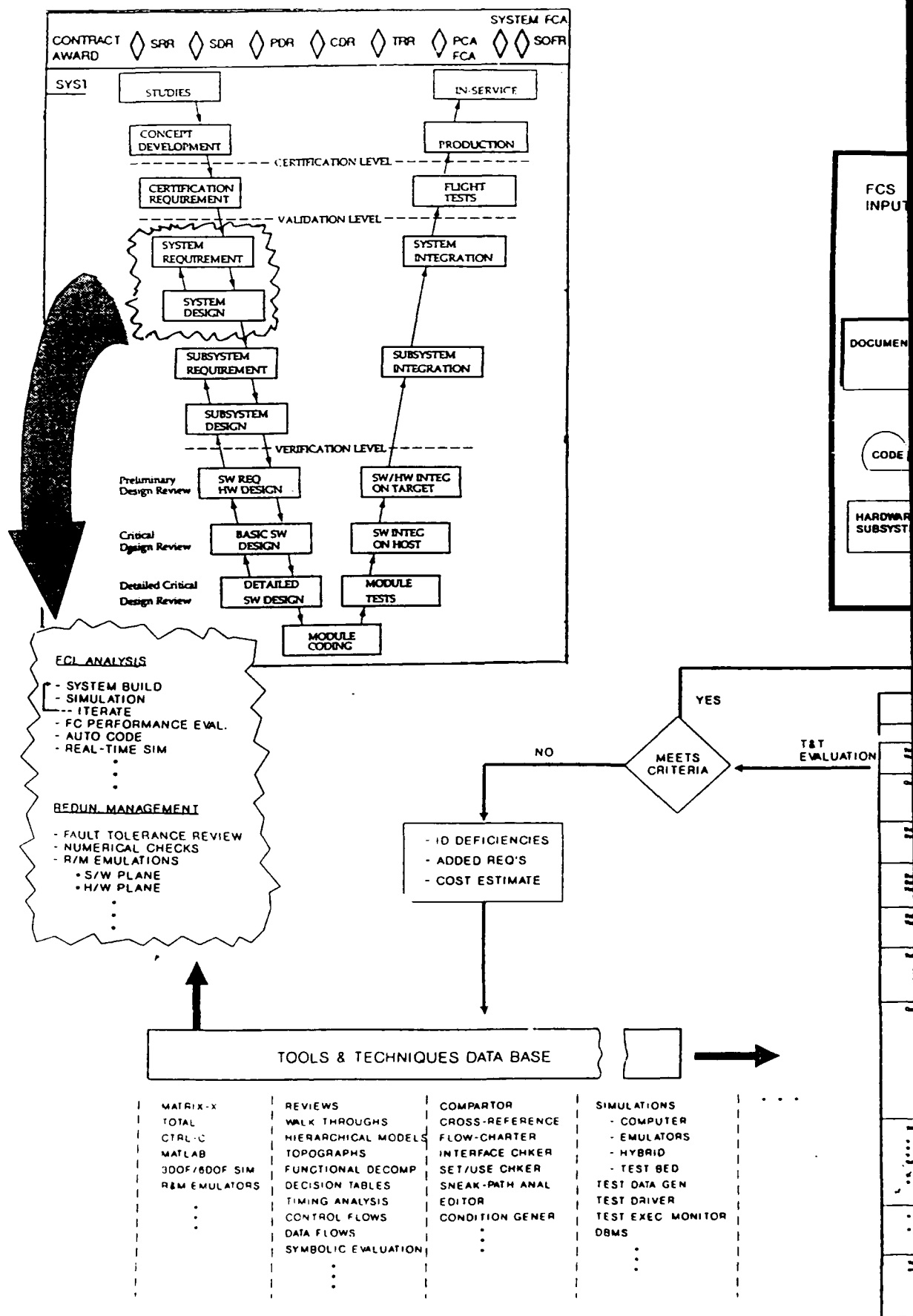




FIGURE 3.5.1-1  
V&V METHODOLOGY  
DEVELOPMENT APPROACH



validation test activities which take place when the development progresses to the point where testing of software and subsystems utilizes hotbench simulators, simulators requiring mainframes for computational support and finally when testing moves into an ironbird test environment.

Review/selection of tools and techniques will be an ongoing activity. This activity will address impacts on V&V requirements as future flight critical systems software designs take advantage of the growth in computational power due to computer advancements and in new design approaches which can be utilized because of this growth. Increases in levels of redundancy for increased safety, use of highly integrated FCS/VMS/PVI to improve mission performance, and use of self-repairing design techniques in FLCS are but a few of the trends which will complicate the verification and validation of FCS software.

The proposed V&V methodology will provide the user a means for early identification of ambiguities and errors in requirements generation and design, yet also provide the means of assessing whether or not the FCS operates as required and is flight worthy.

The specific details of the CAV<sup>2</sup>ES will be presented in the next section.

### 3.5.2 Computer Aided Verification and Validation Engineering System

The FCS V&V methodology design will be implemented in a workstation environment denoted as the Computer Aided Verification and Validation Engineering System (CAV<sup>2</sup>ES), see Figure 3.5.2-1. The CAV<sup>2</sup>ES will provide an environment in which the flight control engineer or software engineer can quickly and easily access and analyze design information and software code, or generate data to verify and validate FCS software. It will allow the engineers to deal with all phases of the development cycle and tackle the problem maintaining a continuity of requirements/design evaluations across these phases.

The Computer Aided Verification and Validation Engineering System will provide the following functions: V&V Executive (VEXEC), Tool Interface Manager (TIM), User Interface (UI), Simulation Computer Interface (SCI), Data Recording

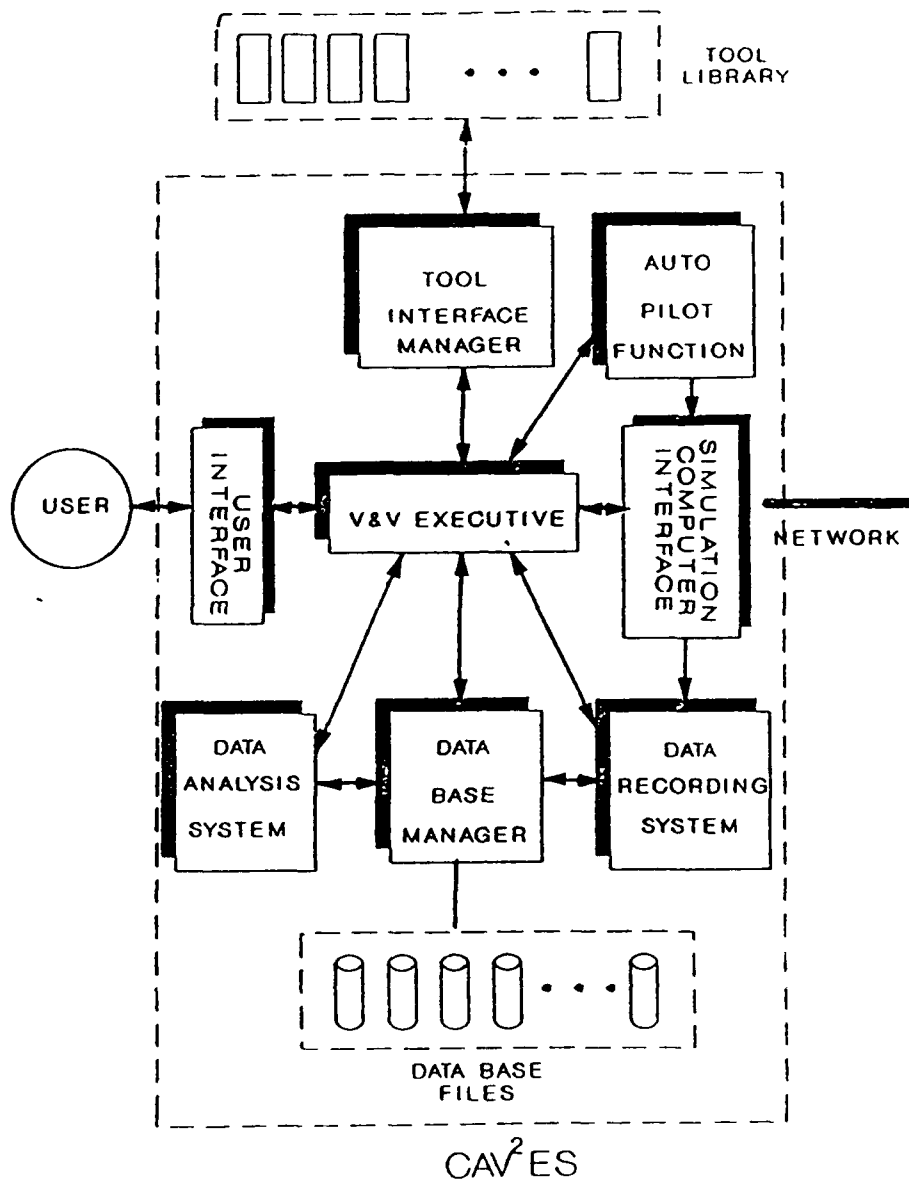


Figure 3.5.2-1 Computer Aided Verification and Validation Engineering System (CAV²ES)

System (DRS), Data Base Manager (DBM), Automatic Pilot Functions (APF), and the Data Analysis System (DAS).

In addition to the CAV²ES primary functions, it also retains its own data base and library of tools. The data base is structured to be able to store and retrieve the data items which are a product of executing the verification and validation analysis tools. The tool library consists of two parts, a generic tools set and an external tools set. The generic tool set is a group of V&V

tools which will perform basic V&V functions on FCS. The external tools set represents user selected or new tools which need to be interfaced through the Tool Interface Manager.

The Tool Interface Manager (TIM) selects which interfaces are used with the selected tools so that the output of the library tool is put into a standard format acceptable to the Data Base Manager. If a new tool (external tool) is to be interfaced to CAV<sup>2</sup>ES, the TIM has an interface build capability which aids the user in building a functional interface which converts data to a format consistent with the existing data base.

The V&V Executive (VEXEC) monitors and coordinates the operation of CAV<sup>2</sup>ES functions. The VEXEC execution involves issuing commands to the Tool Interface Manager, the Simulation Computer Interface, the Data Recording System, the Data Base Manager, the Automatic Pilot Functions and the Data Analysis System. It receives commands and sends responses to the user via the User Interface.

VEXEC is command driven via the User Interface (UI). The user interfaces with the VEXEC by means of commands transformed through the UI. VEXEC assists in selection of: V&V tools, data analysis techniques, and data bases to be used. VEXEC can also assist in start up and shut down of simulations and facilities to be exercised via the Simulation Computer Interface. It can direct the loading of simulation software in simulation computers, and can also control the initialization of the data recording system and other CAV<sup>2</sup>ES subsystems.

For V&V static analysis, VEXEC controls the loading and execution of the selected V&V tools, selected FCS software, design structures and code, and the data analysis and output presentations from the results. For dynamic analysis, VEXEC controls the loading and execution of flight test plans and corresponding test procedures/test cases which comprise them. Execution of the test procedures/test cases can be performed automatically or single stepped. All actions performed by the VEXEC (as a part of static tools execution or dynamic

execution) and all user inputs are recorded in a test execution log which is available on-line for review.

The purpose of the User Interface (UI) is to provide direct access to the various software tools functionalities, while relieving the user of needing intimate knowledge about the software tools as stand-alone systems and adapting to their various styles and syntaxes. This means that a user who wants to obtain a time history plot of data generated by a simulation tool, ACSL for example, does not need to know the particular commands for the tool package for simulation and plotting. However, the UI does not confine the experienced tool user to stay within the UI interface, but provides a direct tool mode in which the user can execute tool commands within the CAV<sup>2</sup>ES environment to perform any simulation and plotting activity allowed by the tool. The UI provides both menu driven and command driven (for the more experienced user) capabilities to the user. The UI provides an open, customizable, flexible environment. The UI is built in a windows environment to provide quick expansion or contraction of backup information, aiding in the verification and validation process. It is also graphically oriented in terms selection of options and in presentation of specification and design information. The functionality of a tool can be accessed via point-and-click mouse operations on icons, menu, and form driven screens. Program structures are presented by schematic designs and network trees; the data displays offer great flexibility in manner of display and in customization.

The Simulation Computer Interface (SCI) is used to communicate to the simulation computers. Communication may take place over serial lines to various devices and over ethernet or bus link. The user may open a terminal window for each of these connections and manually type commands. All commands, along with responses, will be logged and sent to the DBM to be recorded in a test execution log. Other subsystems of CAV<sup>2</sup>ES may also send commands to the simulation computers. All commands indicate if a response is expected. SCI will then pass along the command and wait for the response, if necessary.

The Data Recording System (DRS) will be responsible for recording simulation data (both real-time and non-real-time) and transferring data to the

data base manager. The DRS receives its commands from the VEXEC. Before a test begins, the VEXEC sends a list of commands to be executed (recording script). The start test signal tells the DRS to begin executing the recording script. The abort signal tells the DRS to stop recording and ignore any recorded data. The real-time simulation recording takes place via a link (bus link or ethernet) connected to the simulation computers via SCI. Proper synchronization is critical if a valid set of data is to be recorded.

The Data Base Manager (DBM) serves two purposes. First, to create and maintain data base files and second, to conduct data transactions for other CAV<sup>2</sup>ES subsystems. THE DBM is composed of two processes to serve these purposes: Vexec\_Interface and Build\_Update. The interface process conducts transactions while Build\_Update is used to create and maintain data base files. To assist in performing these processes, a commercially available data base management system, UNIFY, will be used. UNIFY uses a Host Language Interface (HLI) to operate at both of these levels. Briefly, UNIFY's HLI is a library of standard data base management function queries, reports, etc. that can be called from standard Higher Order Language (HOL) programming language statements. The modules and functions of the DBM processes will therefore be written in a chosen HOL.

The Automatic Pilot Functions (APF) subsystem provides the capability for the CAV<sup>2</sup>ES system to send pilot commands to the aircraft flight control system. The APF provides the capability to perform initial condition trimmed (ICT) to specified flight conditions, to provide flight test functions (FTF) for testing of performance parameters (e.g., steps, doublets, sine wave, etc.), to fly fundamental maneuvers, and landing approaches. VEXEC obtains commanded maneuvers from the test procedures via the Data Base Manager. The APF sends these commands to the simulation computers and flight control system via the SCI. The APF uses a transportable auto-pilot model which may be hosted on the simulation computers (mainframes) or in the CAV<sup>2</sup>ES workstation environment, dependent on the particular type of communications link used between the CAV<sup>2</sup>ES and the simulation computers. The communications link must be fast enough to provide realistic commands and feedback for the autopilot to provide

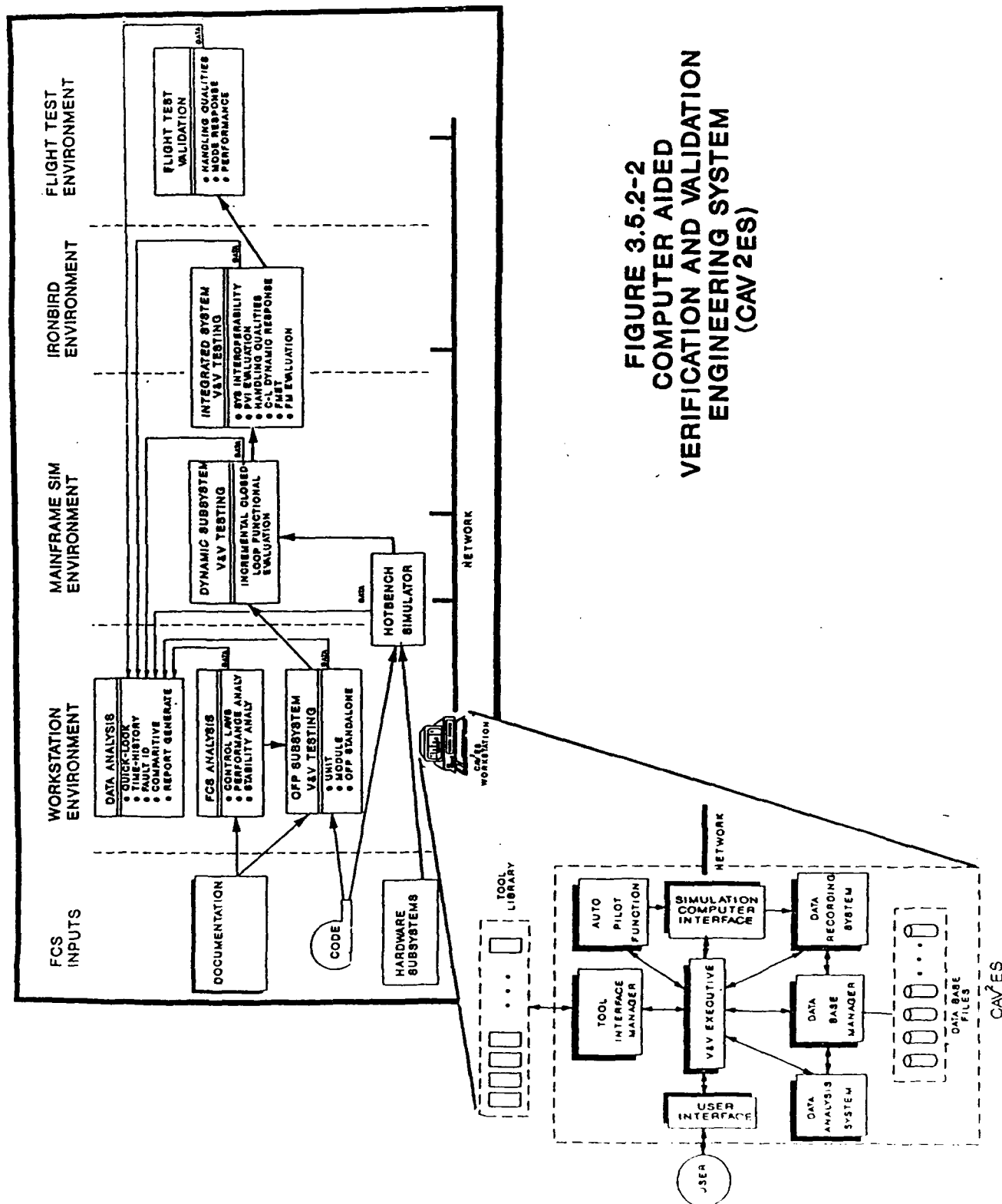


proper control inputs. This provides flexibility in the choice of this communication link from one implementation to the next. There is no intent to provide "pilot modeling" in this module, but simply provide the capability to fly in a controlled manner and to provide controlled inputs which would normally be supplied by a pilot. This module allows elimination of the pilot from many closed-loop tests.

The Data Analysis System (DAS) is designed to provide the validation engineer the capability to examine the data recorded during a test and to perform data reduction techniques on the recorded data. A separate data set is created for each test case executed. The DAS may be used to examine the data in any of the data sets. The recorded data can be displayed both on a dynamic display (bitmapped graphics CRT) and on a hardcopy device. Data displayed during simulation execution will generally be plots of variables as a function of time and mode switches. Post test data analysis can be performed providing performance parameters of flight critical systems. Additionally, the user may specify what variables are to be displayed for a given data set.

The application of the CAV<sup>2</sup>ES in FCS software development and in its verification and validation is shown pictorially in Figure 3.5.2-2. The CAV<sup>2</sup>ES can be used in the early phases of FCS software development efforts to perform FCS analysis and to aid in requirements and design analysis as discussed in Section 3.2. As the FCS development progress, CAV<sup>2</sup>ES provides the analysis tools and techniques to verify requirements and design, to perform OFP subsystem testing at the unit, module, and subsystem levels. Software tools can be hosted to aid the user in evaluation of system reliability. Development of analytical techniques are becoming available to aid in performing this task. The fault tree approach is one tool/technique which will be evaluated to include in the tool library. Other techniques will also be reviewed.

The CAV<sup>2</sup>ES provides an environment in which the V&V engineer can use the inputs of previous verification efforts including requirements analysis, design analysis, and code analysis to quickly generate test cases for execution of the FCS software. Within this environment, the process of generating test cases



can be eased by providing help or advisory instructions for testing of specific subsystem/system segments. Also, test cases previously used for achieved specified test objectives can be quickly pulled from the test case, data base and used as examples. These examples can then be quickly tailored to meet specific design test specifications, if required. Following this, CAV<sup>2</sup>ES can provide an effective "test directors" workstation which uses a data base of proven test procedures to perform hotbench testing, dynamic subsystem testing and finally integrated system validation testing in an ironbird environment. CAV<sup>2</sup>ES not only provides the test directors control capabilities over the testing, but also supplies data reduction and analysis tools to analyze the test results. Finally, the CAV<sup>2</sup>ES can be used to support flight test by examining (real-time simulation) flight scenarios prior to flight test to predict flight test results. These same analysis tools can also be used to reduce flight test data and compare them to predicted results.

### 3.5.3 CAV<sup>2</sup>ES FCS V&V Capabilities

Capabilities to be included within the CAV<sup>2</sup>ES environment will provide the flight control/software engineers the capability to assess the FCS software design in terms of performance, stability, and redundancy management analysis. It will provide both static and dynamic code analysis tools for verification and validation of flight critical systems code. It will also provide the capability to perform quick-look analysis of generated data. It will provide the means to verify and validate FCS software through control of real-time simulators driven by proven test procedures/test cases. Specific V&V capabilities and techniques/tools to be applied will be presented below in an order which would correlate to the flight critical system software development.

#### 3.5.3.1 Aircraft Flight Critical Systems Analysis

The CAV<sup>2</sup>ES will provide the capability to perform verification and validation testing of FCS software by evaluating the adequacy of the control laws with respect to performance and stability and to evaluate system mechanization with respect to redundancy management, timing and bus loading. To perform these analysis, 3 general types of flight control analysis tools

will be used: a linear analysis and design tool, a nonlinear simulation tool, and a system level emulator. Figure 3.5.3.1-1 depicts these tools. The non-linear simulation will be used to evaluate large amplitude characteristics of digital FCS and to provide state space models for linear analysis. The linear analysis tool will be used to evaluate stability and performance margins, and sample data properties of the closed-loop control system. The system level emulator will be utilized to verify that the implemented code represents the system design with sufficient accuracy to meet system performance requirements. Outputs of the system level emulator can be used to provide inputs for rigorous exercising of operational flight code, when available, on an instruction level emulator of the target flight computer. A more detailed discussion of each of these types of tools follows.

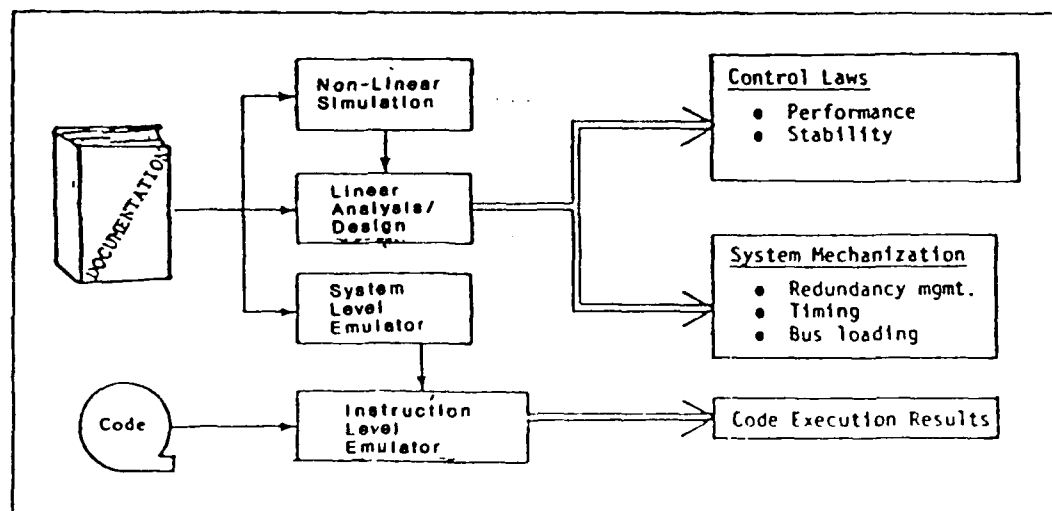


Figure 3.5.3.1-1 Performance, Stability & Mechanization Tools

## Control Law Analysis

The control law analysis must insure that digital FLCS meets the flying and handling requirements within the maneuvering flight envelope of the aircraft, as specified by the system specification and the MIL-SPECs. This effort includes control law specification assessment, non-linear simulation, and control law evaluation.

Usage of the nonlinear simulation and linear analysis tools for control system analysis and evaluation is depicted in Figure 3.5.3.1-2. A general purpose non-linear aircraft simulation program incorporates specific aircraft characteristic through user-defined modules for the aerodynamic forces and moments, the propulsion system, the control system, etc. It can be run to trim the aircraft for any desired flight condition, to generate linear state models for the trimmed flight condition, and to generate time history responses for user-defined inputs representing commands or external disturbances.

Commercially available linear analysis and design tools provide a comprehensive interactive control design and analysis software language system including state-of-the-art primitives in classical and modern control synthesis, matrix analysis, dynamic system analysis, parameter estimation, and graphical presentation. System analysis tools that are significant for investigation of digital flight control systems are continuous to discrete transformation, time and frequency responses, stability and performance robustness computations, MIL-F-8785C analysis report generation and computation of control requirements.

## Performance and Stability Analysis

There are two characteristics of the DFICS which are critical to aircraft safety of flight. These are the aircraft response to pilot and turbulence inputs, and the ability of the flight control system to cope with deviations from the model on which it is based. Of particular importance for this

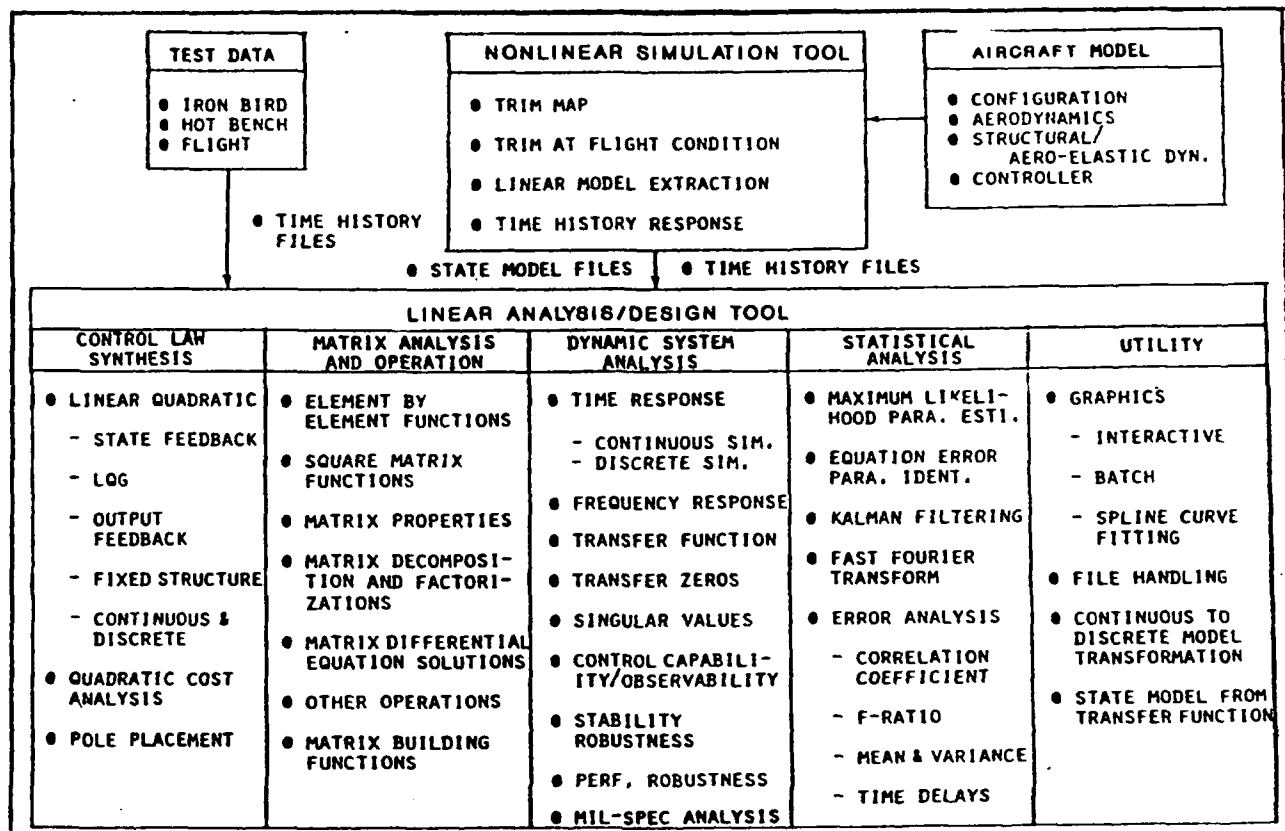


Figure 3.5.3.1-2 Nonlinear Simulation and Linear Analysis Tools

analysis is the effect of discretization on the stability and performance robustness of the aircraft. As was discovered in the AFTI/F-16 DFIC design, for the low frequency components of system response discretization of the analog design may be used. However, the narrow structural notch filters require direct discrete design in order to avoid problems arising from aeroservoelasticity and the warping of filter characteristics. Algorithms for performance and stability robustness computations are available for evaluation

of FLCS software design. Table 3.5.3.1-1 outlines a number of tasks which can be performed to evaluate the stability and performance of digital control laws using linear analysis/design tools.

Table 3.5.3.1-1 Control Law Analysis Tasks

DISCRETIZATION ANALYSIS

- discretization errors
- equivalent transfer function error

STABILITY ANALYSIS

- continuous and discrete closed loop eigenvalues
- multiloop phase and gain margin based on singular value analysis

DISTURBANCE REJECTION ANALYSIS

- transient response for step wind gust
- RMS tracking accuracy for random turbulence
- disturbance rejection robustness based on singular value analysis

CONTROL REQUIREMENTS

- control surface position and rate for deterministic and stochastic inputs

PERFORMANCE ANALYSIS

- transient response for pilot inputs
- tracking performance robustness based on singular value analysis

System Mechanization Analysis

The system mechanization analysis verifies that the code implements the system design with sufficient accuracy to meet performance requirements. To perform this analysis and validation, the requirements contained in Software

Mechanization Documents, together with the aircraft dynamics would be implemented in a system level emulation structure. The emulation can then be utilized for redundancy management analysis, timing and bus loading analysis.

The structure of a typical system level emulator is illustrated in Figure 3.5.3.1-3. Use of the emulator requires that the DFLCS requirements/design be implemented (modeled) and that the appropriate aircraft dynamics be added. This type of emulator can be used to analyze the correctness of module logic and functions, bus loading, timing and redundancy management, as well as analyzing the operational capabilities of a system and its conformance to system requirements.

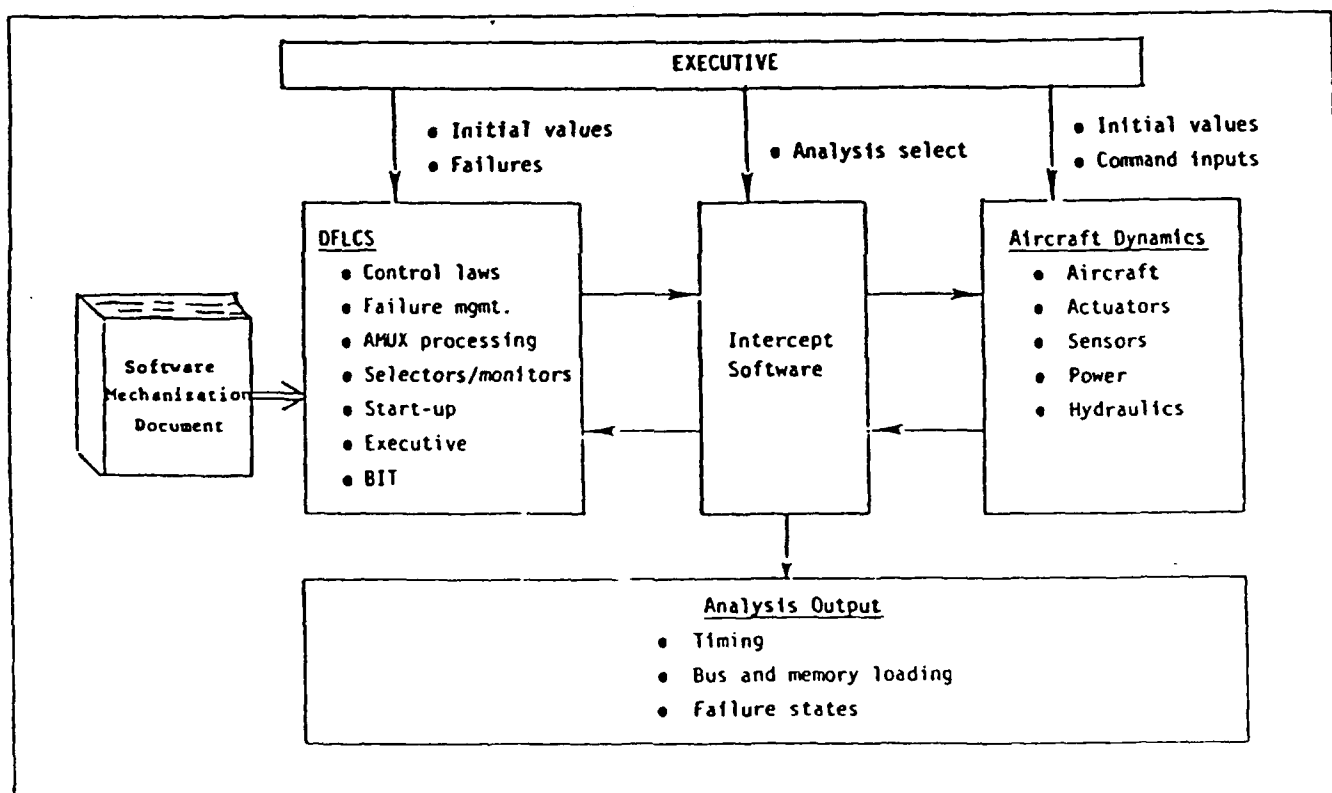


Figure 3.5.3.1-3 System Level Emulator



The executive controls the simulator and sets failures, initial values, aircraft flight conditions, and selects the type of analysis to be performed. The DFACS emulation consists of modules with processing modes and communication links of sufficient detail to accommodate the desired component failure conditions to be injected by the user.

Typical questions which are addressed through use of an emulator are:

- o Are there deficiencies in the Mechanization Document?
- o Does the detailed software design faithfully represent the Mechanization Document requirements?
- o What happens in the case of certain failures which appear simultaneous to the software (e.g. latent failures)?
- o Is the design susceptible to particular cases of interchannel skew?
- o Is mode transition accomplished smoothly among the asynchronous channels? (same for failure detection and reconfiguration, and for failure reset).
- o Do all tasks get serviced by the executive within timing requirements?
- o Is there adverse coupling between the Failure Manager and Control Laws, e.g., are failure latching and control law configuration handled smoothly by the flight control executive?
- o What is the minimum acceptable time to detect and isolate various types of failures? Is that time requirement met by the design?
- o What are typical inter-channel differences and are threshold selections tolerate of these?
- o Does the persistence-counter design provide for a reasonable trade of resistance to false alarm for speed of detection?

#### 3.5.3.2 FCS Software Design Verification

Software design analysis activities are directed at verifying:

- o the allocation of system and software requirements to software components
- o the adequacy of the design to meet the requirements
- o compatibility of the software with both external and internal interfaces.

The primary aim of the design verification is to confirm that the recommended design will perform the function specified in the Requirements Specification.

The approach to software design verification is illustrated in Figure 3.5.3.2-1. The approach is to evaluate the system requirements documentation, progressing through development outputs and approved baseline design documentation. Analysis of the system and software requirements is the initial step of design verification and is normally addressed as part of the FCS and control law analysis discussed in Section 3.5.3.1. The system specification, Part I design specification, interface compatibility documents, trade studies, and other documents provide inputs for verification of the requirements allocation. Requirements are analyzed to confirm that: all functional, interface and test requirements are completely specified in quantitative terms; requirements are logical, consistent, testable, and traceable; all data base and data requirements are clearly stated; all equations have been scientifically verified; and timing and sizing estimates have sufficient margin for growth. Upon reviewing the requirements documentation, and the FCS software development and management plans, a requirements compliance checklist may be used to verify that the given design adequately addresses all system requirements. Example requirements compliance checklist and software design compliance checklist are presented in Figure 3.5.3.2-2.

The next step in the verification is to confirm the technical adequacy of the design. This process is to verify that the total design has been expressed in writing and that adequate analysis, simulation and evaluation have been performed to evaluate the design as to risk, expected performance, cost and reliability. The design should address performance capabilities, system and software architecture, operational sequences, information flow, timing and other parameters. Design elements are analyzed for ambiguities, maintainability, expandability adequate decomposition of design components ensuring a top-down design and that testability and maintainability considerations are embedded within its structure.

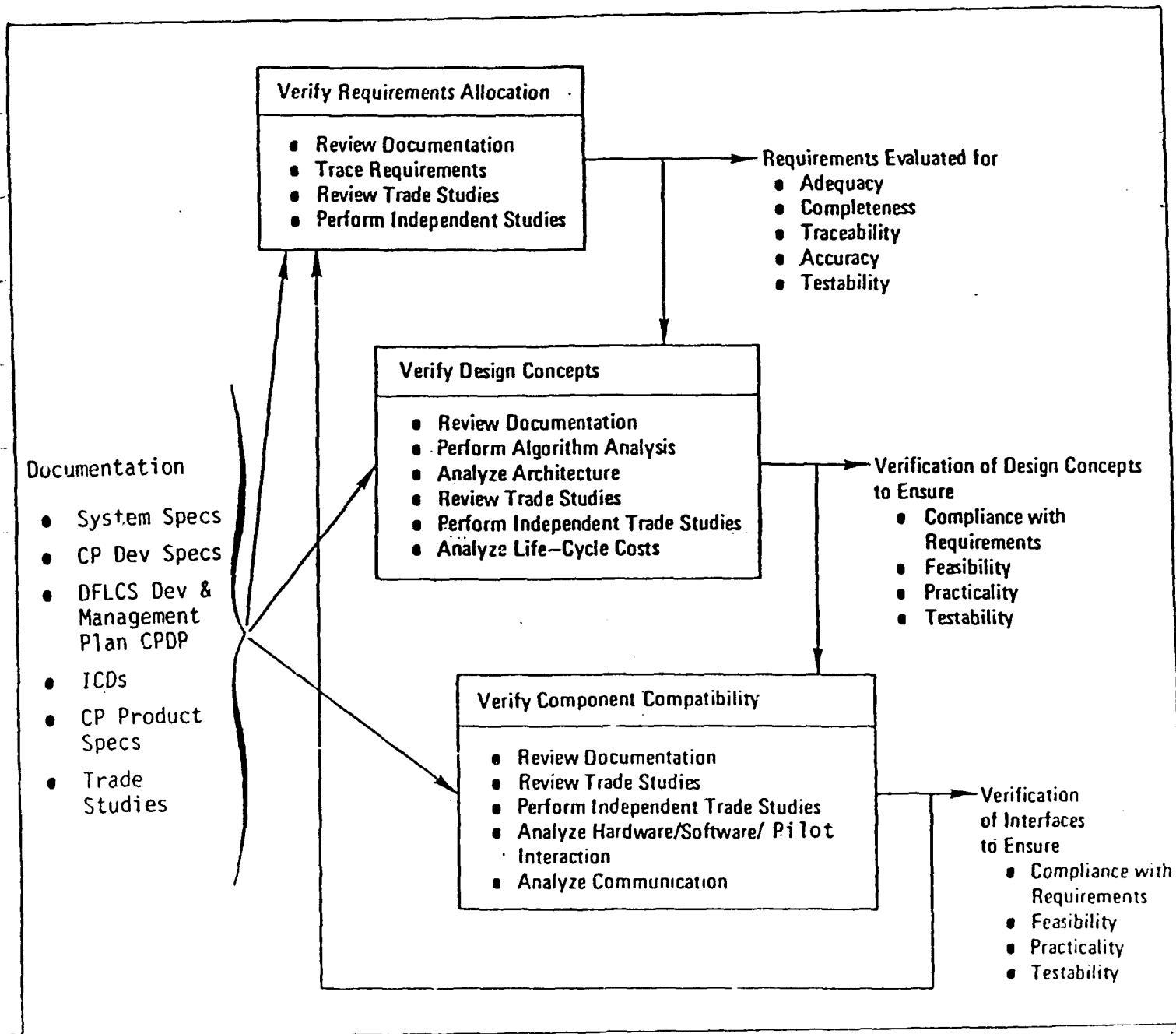


Figure 3.5.3.2-1 Approach for Performing Software Design Analysis

SOFTWARE DESIGN VERIFICATION CHECKLIST	
CPC: CONTROL LAWS	
FUNCTION: GAIN SCHEDULER	
	YES NO
• Have all software requirements been addressed in the design and is there traceability?	X
• Do all the equations, algorithms, and input/outputs correctly mechanize requirements?	X
• Is the data base fully defined and is its architecture (structure and access methods) fully compatible with capabilities and their complex images defined?	X
communications and interface the Part I Development Specification to in the design?	X
ible with the hardware and external established in ICD's and the Part I actions?	X
sizing budgets established at the	X
signs fully realize overall requirements, operation, growth, maintainability,	X(1)
ed enough to begin coding?	X
timing and sizing margin.	X
that the specifications define digital FLUX that is to that of the AFCS. The digitization introduces rad/s, that increases up to the Nyquist frequency of	
side smooth transitions for slow changes of dynamic velocity at high angle of attack causes a 4 Hz	
in the actuator commands for the horizontal tail, attenuated by the actuator dynamics. The effect of o-elastic and aero-servo-elastic characteristics of with BCT's simulation.	

SOFTWARE REQUIREMENTS VERIFICATION CHECKLIST	
CPC: EXECUTIVE	
	YES NO
• Are all functional, interface, and test requirements completely specified in quantitative terms?	X
• Are there any potential problem areas in fulfilling the requirements?	X (1)
• Are the requirements logical, consistent, testable traceable, and understandable?	X (2)
• Are all input, output, and processing requirement identified and specified for each function without ambiguity?	X
• Are all hardware and software interfaces identified?	X
• Are the data base and data requirements clearly stated?	X
• Are acceptance criteria specified for each requirement?	X
• Have the equations been scientifically verified?	N/A
• Are the objectives and stages of testing described?	X
• Do timing and sizing estimates have sufficient margin?	X
1. The discussion of the Primary Mode Executive in the Computer Program Development Specification needs to be re-written for clarification relative to the status of incomplete tasks, and the setting of the VDT inhibit indicator.	
2. The "task scheduling request" is an input to the Executive but is not presented as such in the Development Specification.	

Figure 3.5.3.2-2 Example Requirements and Design Compliance Checklists

Particular attention is directed to design strategies which involve external interfaces such as the hardware/software/pilot interaction. The design is examined to ensure that all external interface requirements have been addressed.

As can be discerned from the description of the design verification tasks, much of this effort involves manual review of requirements and design specifications, requiring methodical and diligent attention in matching requirements to design and in evaluating designs. It is in this area where the application of CASE tools discussed in Section 3.5 can provide benefits in verification. In general, CASE tools leverage the requirements analysis and design specification phases of the software development cycle while more traditional tools are more applicable in the software implementation phase. CASE tools can be applied from an independent viewpoint to evaluate requirements/design relations. They can provide structured analysis of these designs/requirements relations to ease the evaluation process of the analyst. CASE tools can yield tremendous benefits in revealing many requirements (and surprises) before implementation begins. Some CASE tools also provide reverse engineering capabilities which will take implemented software back to graphic structured design representations. In this form, verification of design to design specification and traceability of requirements to design can be performed more easily.

For this effort, Frontier will evaluate currently available tools and incorporate one or two of the most applicable tools. Teamwork, Battlemap and Tracebuilder II are examples of CASE tools that are applicable in this design verification task.

#### 3.5.3.3 FCS Software Code Verification

The objective of this software code verification is to evaluate the code for technical correctness, efficiency and adequacy. Tools and techniques to aid in performing software code verification are many and varied. The general classes of tools and techniques were discussed in Section 3.5. The code

analysis will utilize tools and techniques that allow for test repeatability, since multiple versions/updates of the code are a natural part of any FCS software development effort. The code analysis will emphasize the software interfaces and sequencing logic. Past experience shows that these areas tend to be very error-prone. The code will be examined using both static analysis and dynamic execution analysis. Static analysis provides statistics on syntax, structural relationships, and cross-references, while dynamic execution analysis will allow evaluation of actual code execution results. The code will also be examined for efficiency. Areas of code that have high utilization will be identified with a timer analyzer tool and then will be further scrutinized for efficiency in coding to minimize run time. Routine and module size is another area which must be monitored. Unplanned trade-offs often occur when sizing and timing constraints reach their limit.

Specific static analysis techniques which will be initially implemented will be selected as a part of the Phase II effort. However, capabilities to perform software sneak analysis and to perform instruction level emulator code execution are examples of static and dynamic analysis tools, respectively. Both provide a strong code verification tool base and are planned for implementation in CAV<sup>2</sup>-ES. A description of the use of these follows.

Software Sneak Analysis (SSA). SSA methodology is based on the development of topological network trees which provide a clearly understandable functional representation of FCS software performance, and which are useful throughout the life of the FCS software. Software sneak analysis outputs include a comprehensive, understandable software network tree database. Instead of basing the development of network trees strictly on NASA developed techniques or other sneak methodologies, FTI builds a network tree database from hierarchical models in a manner that clearly reflects a program function, in a format understandable to hardware, software, and system engineers. This database is one of the major benefits offered by a software sneak analysis approach. Initially each line of executable source code is converted to one or more models as shown in Figure 3.5.3.3-1. The modelling process is hierarchical. For instance, in a background executive, the main program may be

represented only as an impedance. In lower level trees, this model will be expanded to include all execution paths.

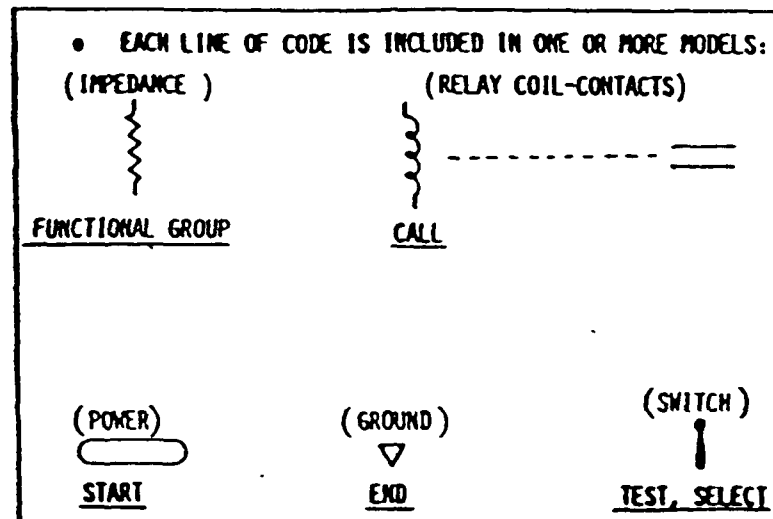


Figure 3.5.3.3-1 Models for Code Representation

The next step in the SSA process is to use pathfinding programs in order to fully trace all possible program execution paths. The programs assist in connecting all source code hierarchical models and displaying them in the software network trees. A unique SSA tool--hierarchical data cross references--are then generated to assist in determining sneak data paths by helping trace data flow across all module interfaces.

Sneak paths, sneak indications, and sneak labels are determined through topograph identification and clue application techniques. Sneak timing problems are discovered through interrupt sequencing and operations analysis. Figure 3.5.3.3-2 illustrates the six basic topographic patterns common to all languages. Each network tree is composed of one or more of these topographs. Each pattern (topograph) has an associated clue list which is used by the analyst to alert him to potential sneak conditions.

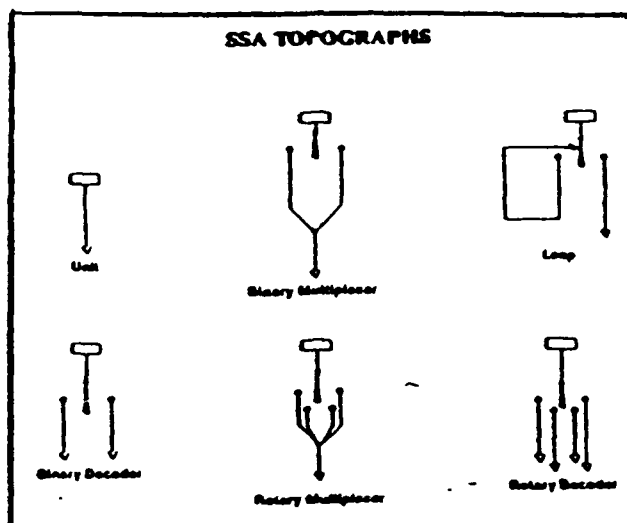


Figure 3.5.3.3-2 Software Sneak Topographs, Common to All Languages, Alert the Analyst to Possible Sneak Conditions

In addition to applying topograph specific clues, a list of application clues are used to help determine sneak conditions. An example of sneak conditions is given in Figure 3.5.3.3-3.

The network tree modelling technique is specifically designed to provide maximum visibility of code function so effects of changes on overall system function can be easily determined.



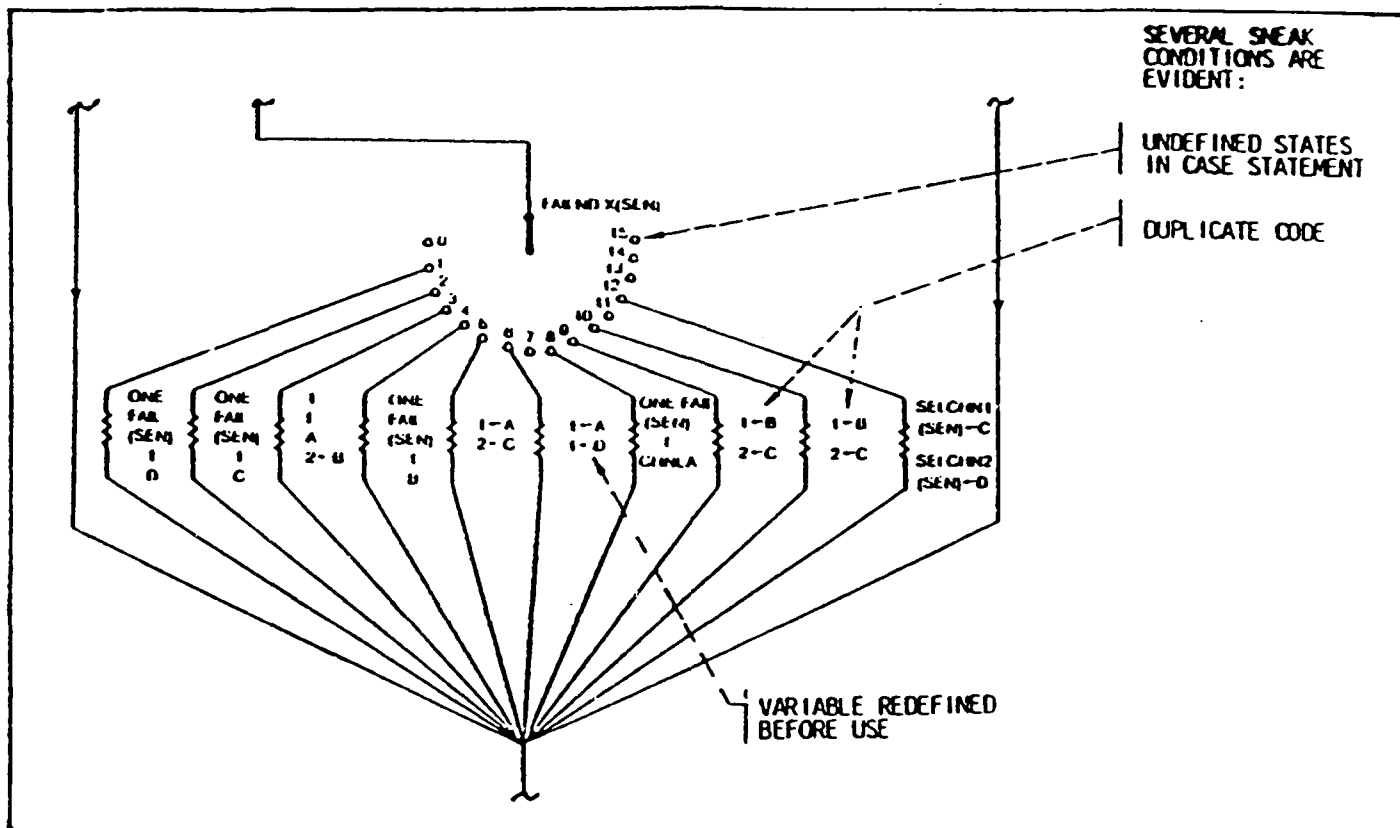


Figure 3.5.3.3-3 Examples From F-16 DFCS SSA Program

### Instruction Level Emulator

To evaluate program correctness, the FCS software will be executed and its response to given stimuli will be assessed against acceptable limits through instruction level emulation. The 1750A simulator is an example of an instruction level emulator which will be employed to execute the software for

two levels of correctness testing. The basic level of correctness testing will exercise all FCS software in a routine-by-routine bottom up fashion. Emphasis here will be on inputting nominal, limit and erroneous data into the routine and evaluating the output for acceptable content and/or arithmetic correctness. Instruction paths will be traced, aiding in the identification of dead code, and in calculation of timing estimates for each routine. As bottom up correctness testing continues, the interfaces between routines will be exercised with regard to control and data passing. The second level of correctness testing will be functional. Those software implemented functions identified as critical and/or suspect from the design analysis effort, software sneak analysis, or other techniques will be executed extensively on the 1750A simulator. The function's response to the test data will then be evaluated against acceptable limits. For both levels of correctness testing, the HOL compiler and linker will be used to allow test drivers, stubs and data extraction hooks to be linked in with the FCS software in order to augment emulation testing capabilities.

#### 3.5.3.4 Stand Alone and Dynamic Subsystem Verification and Validation

The engineering and formal system and software testing utilizes a bottom-up philosophy. Testing starts with a lowest level code (unit code) and progresses upward to system-level testing. Verification of results is performed at each level before progressing to the next level. Test plans and procedures are prepared for each level of formal testing and test results are documented. Problems or discrepancies detected during any phase of testing is documented, investigated, and acted upon.

The stand alone verification and validation is the first level of testing involving actual flight hardware. The objective of this testing is to insure that the Operational Flight Program (OFP) is functionally correct. Ideally, this testing should be automated to the maximum extent possible to allow for rapid retesting when future updates are made. The testing verifies requirements specified in Software Requirements Specification, provides open-loop functional evaluation, and supports computer software configuration/hardware integration testing. This test environment uses engineering test

stations containing models to drive different subsystems containing flyable hardware tests. Test files provided by CAV<sup>2</sup>ES are used to drive tests. The results of these tests are then compared to predicted results and test reports are automatically generated.

The next step in the V&V testing is to integrate individual "subsystem" tests to provide dynamic rather than static inputs between tested subsystems. This can be done to different levels as illustrated in Figure 3.5.3.4-1. Flyable subsystems are incrementally integrated via engineering test stations. This type of testing provides added testability of integrated subsystems and reduced hotbench or iron-bird simulation requirements. CAV<sup>2</sup>ES will be used to communicate to simulation computers and subsystems to drive individual and integrated subsystem tests. The Maneuvering-simulation Validation Automation (MVA) system currently under development by FTI for SAAB-SCANIA will provide just this type of capability. The design structure of CAV<sup>2</sup>ES can incorporate many of the functions in the MVA system.

#### 3.5.3.5 Integrated System Verification and Validation (ISVV)

Integrated system verification and validation testing provides a closed loop test environment that simulates the dynamic behavior of the air vehicle. Actual flight hardware is used where practical to verify closed loop dynamic responses. This test environment is used to perform pilot-in-the-loop handling qualities evaluation, pilot vehicle interface, and failure modes and effects tests. This test configuration is represented in Figure 3.5.3.5-1. The CAV<sup>2</sup>ES will be used to provide a test directors work station environment for conduct of these types of tests.

## DYNAMIC SUBSYSTEM V & V

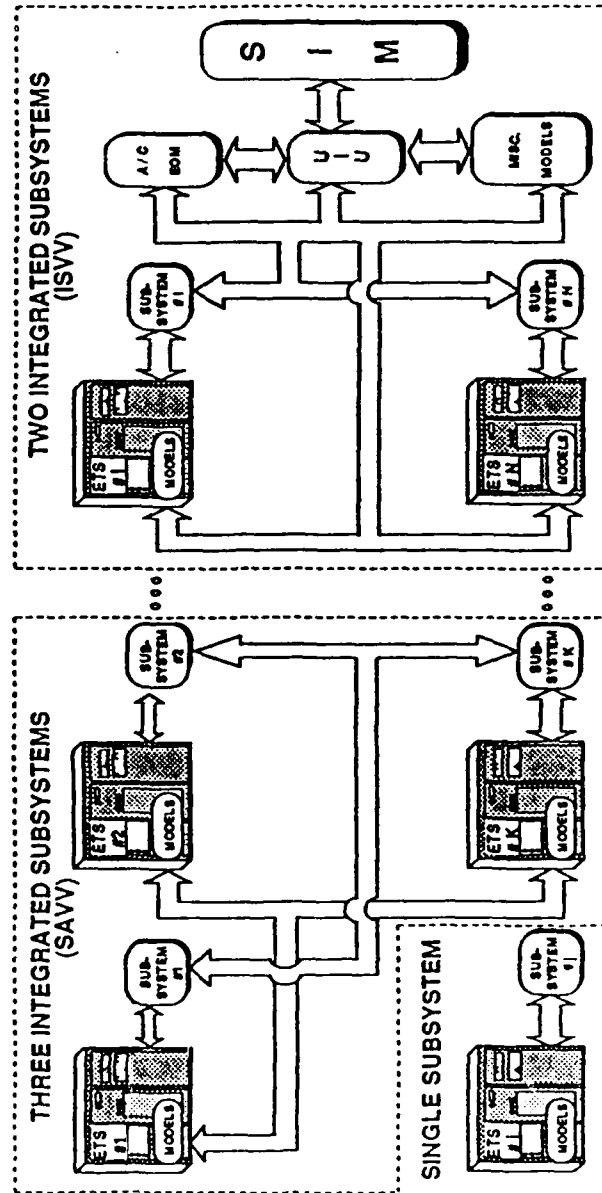


FIGURE 3.5.3.4-1 SUBSYSTEM VERIFICATION & VALIDATION

The entire system test and evaluation process is represented in Figure 3.5.3.5-2. This includes the FCS and software along with airplane subsystem tests. The culmination of all these tests is flight testing.

In summary, the CAV<sup>2</sup>ES is an integrated V&V tool set to be used throughout FCS development phases. The specific tools and techniques discussed for implementation will provide the user the capability to analyze the FCS software design and code, to allow him to easily generate test cases, and to execute these test cases in various test environments. It provides the capability for performing reliability analysis through user selected techniques. The user is able to call on analysis data generated from different V&V analysis phases as input to his current task. It is usable by developers, V&V organizations, and research groups. Its workstation provides the user with a powerful computational environment which can utilize many of the more sophisticated FCS design and analysis tools. CAV<sup>2</sup>ES provides a CASE type working environment, user friendly interfaces, and provides hooks for addition of future tools. It provides means for interfacing and driving real-time simulations and offers an environment to approach automatic generation of test cases while also aiding automation of real-time testing. CAV<sup>2</sup>ES is an evolutionary system which accommodates new tools to meet the growing user requirements in verification and validation of flight critical systems software.

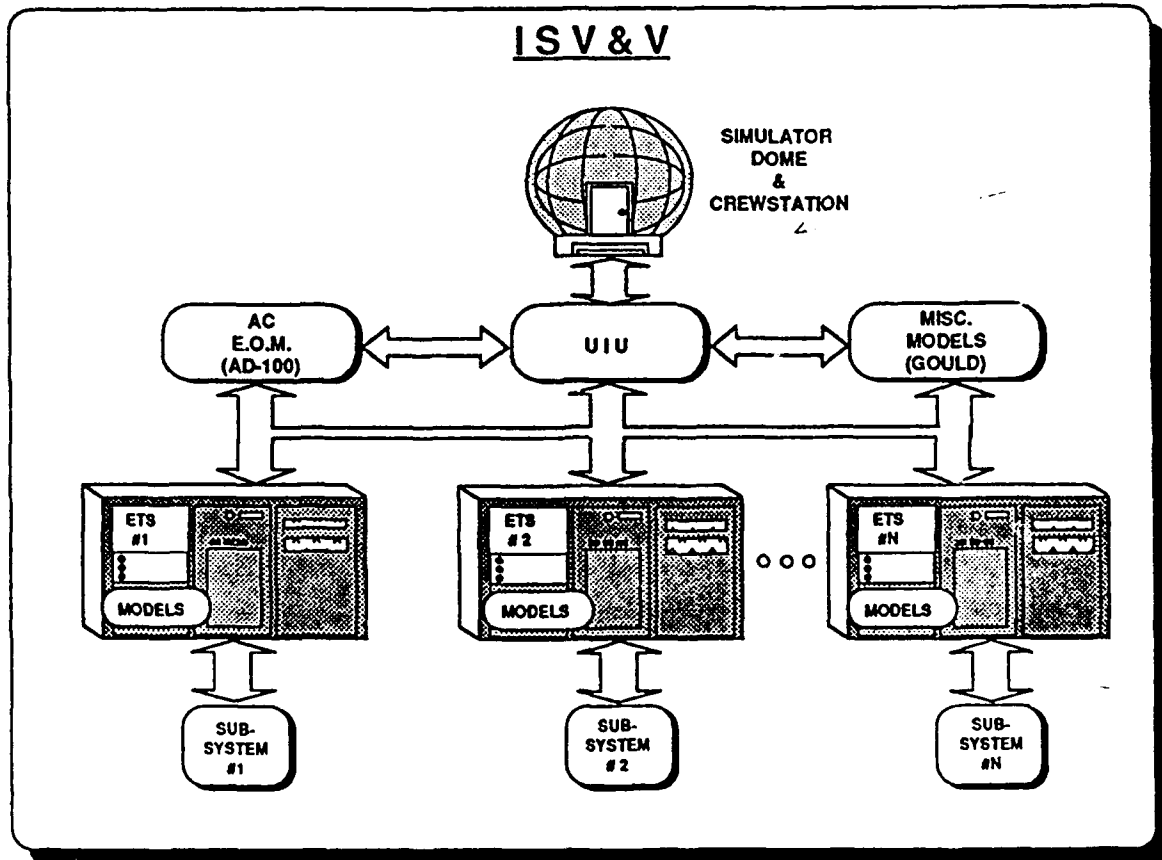


Figure 3.5.3.5-1 Integrated System Verification and Validation

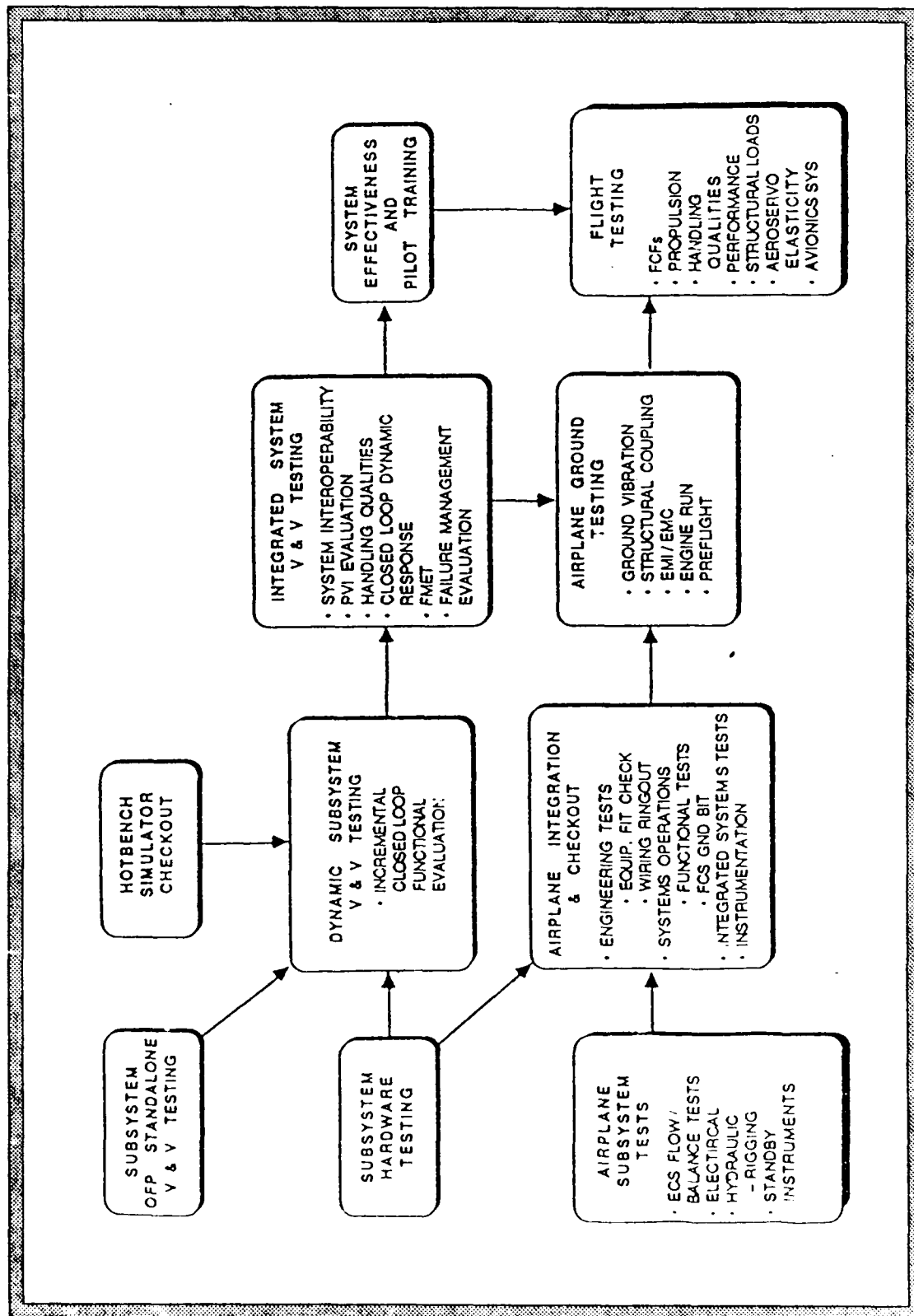


FIGURE 3.5.3.5-2 SYSTEM TEST AND EVALUATION

Appendix

Data Gathering Checklists



QUESTIONNAIRE TOPICS

TYPES OF SOFTWARE DEVELOPED

DEVELOPMENT PROCEDURES/METHODOLOGY

MANAGEMENT OF DEVELOPMENT EFFORT

FLIGHT CRITICAL SYSTEMS DEVELOPED BY THESE METHODS

DEVELOPMENT LANGUAGES

SOFTWARE TOOLS AND TECHNIQUES

STANDARDS, PRACTICES AND GUIDELINES

VERIFICATION AND VALIDATION TECHNIQUES

QUALITY ASSURANCE TECHNIQUES

IMPACT OF GOVERNMENT STANDARDS

GENERAL FACILITY DATA

FLIGHT CONTROL SYSTEMS REQUIREMENTS ISSUES

GENERAL PROBLEM AREAS IN DEVELOPMENT

SYSTEM DEVELOPMENT ERRORS AND STATISTICS

#### TYPES OF SOFTWARE DEVELOPED

Flight Control  
Avionics Architecture  
Engine Control  
Command, Control, Comm  
Analytical Models  
Simulation  
-  
-

#### DEVELOPMENT PROCEDURES/METHODOLOGY

Waterfall  
Spiral  
Prototyping  
Hierarchical Phase Model  
-  
-

#### SOFTWARE DEVELOPMENT PHASES

System Requirements  
Systems analysis and design  
Software analysis and design  
Coding and checkout  
Software integration testing  
Software/System qualification  
Software Maintenance

#### REVIEWS AND TESTS

Stage	Reviews	Tests
1. Requirements	Systems requirements review	--
2. System design	Software concept	--
3. Software design	Preliminary design review	--
4. Coding	--	Module tests
5. Integration	Critical design review	Integration tests
6. Qualification	Qualification Audit or functional Configuration audit	Validation tests on operational hardware
7. Installation	Physical configuration audit and Formal qualification review	Validation tests On iron-bird simulation
8. Maintenance	Change reviews	Re-validation tests

## MANAGEMENT OF DEVELOPMENT EFFORT

- o Background of software designers: control engineers? software engineers?
- o What aspects of software project management functions are used:
  - Management
    - Project Planning
    - Project Control
    - Project Communications
  - Documentation
    - Engineering Documentation (2167, PDL, etc.)
    - Formal Management Documentation (2167, SDP, S/W Dev Plan, CM Plan, QA Plan
    - Informal Documentation (reports, guidance/policy papers, minutes, etc.)
  - Configuration Management
    - Baseline Identification (Tech descrip of all S/W items)
    - Control & Tracking of S/W Access and Change
    - Control of S/W releases

## FLIGHT CRITICAL SYSTEMS DEVELOPED BY THESE METHODS

F-16 DFCS  
F-15 E  
AFTI/F-16  
F-15 STOL  
X-29  
F-18  
PAGUS  
X-31  
F-117

## DEVELOPMENT LANGUAGES

Ada  
Jovial J73 (MIL-STD 1879B)  
Assembly  
CMS-2  
-

## SOFTWARE TOOLS AND TECHNIQUES

### Proprietary Tools

-

### Flight Control Analysis Tools

- Matrix-X
- CNTRL-C
- MATLAB

-

### CASE tools

-

-

-

### TECHNIQUES

Abstractions and hierarchies to reduce complexity: abstractions such trees are used to make the design simple and clearly defined.

Checkout (debug) testing: function/module testing before integration

Constructive design approaches: (eg. formal design language)

Critical design Review: oral demonstration of detailed design

Data flow diagram, structure chart: used in preliminary design

Descriptions or documentation

Design guidelines, test guidelines, & coding guidelines

Design standards, coding standards

Functional capabilities list: module description of functions to perform

Integration testing: code test after modules are assembled; I/O struct.

Organization as finite automata

Qualification audit

Singularities and extremes testing

Symbolic execution: performed on special functions such as mode logic

Systems concept review: oral demo of initial concepts, trade-offs, etc.

Validation testing: final demo in simulation environment

## TOOLS

Accuracy analyzer - analyzes numerical calculations for red accuracy

Circular reference checker - modules calling each other

CAE

Code comparator - differencing between versions

Cross-reference checker - calling of modules; external variables called

Data bases analyzer - module accesses to data bases; unused elements

Data Flow Pathing - trace execution sequence for variable(s) in flow

Documentation & constructions - Auto documentation; Consistent data pool

Editor - Analyze/extract information/relationships from source programs

Emulations - System level model generated from requirements, not design

Flow charter - show logical construction of program

Formal Languages - Structured and rules.

Interface Checker - Check Range, limits, scaling of variables

Module Invocation Tree - Establishes call hierarchy with system

Program flow analyzer - statistics on usage; estimate execution time

Set/Use checker - Checks for variables: set, not used; & used before set

Simulations - Test characteristics, algorithms, functions, performance

Sneak-Path Analyzer - Looks for unexpected paths

Symbolic evaluator - reconstructs equations relating output to input

Test data generator - produces test cases to exercise the system

Test driver - controls the execution of a program

Test execution monitor - collects data and compares to expected results

Test record generator - analyzes, reduces, and formats results

Theorem prover - axioms used prove assertions stated for a path

Timing analyzer - monitors/records run time of functions and routines

Units consistency checker - variable expressions (units) checked.

Unreachable code detector - looks for code which cannot be executed

Specific Static Tools	General Static Tools	Dynamic Tools
Circular reference checker Code comparator Consistency checker Cross-reference checker Data base analyzer Flow charter Interface checker Program flow analyzer Set/use checker Standards checker Units consistency checker Unreachable code detector	Accuracy analyzer Assembly code verifier Assertion checker Documentation and construction systems Formal languages with syntax analyzers <ul style="list-style-type: none"><li>• Requirements</li><li>• Specifications</li><li>• Program design</li><li>• Program code</li></ul> Sneak-path analyzer Symbolic evaluator Theorem prover Verification condition generator	Simulations <ul style="list-style-type: none"><li>• Computer</li><li>• Hybrid</li><li>• Test bed (iron bird)</li><li>• Monte Carlo</li></ul> Test data generator Test driver Test execution monitor Test record generator Timing analyzer

## STANDARDS, PRACTICES AND GUIDELINES

- Design standards
- Coding Standards
- Documentation Standards
- Engineering Development Standards
- 
- 

## VERIFICATION AND VALIDATION TECHNIQUES

- Review & Walkthrough
- Units Consistency Checks
- Data Flow Charts
- Interface Format Checks
- Decision Tables
- Set/Use Checks
- Type Consistency Checks
- Timing Test/Analysis
- Numerical Accuracy/Precision Analysis/Test
- Symbolic Evaluation
- Testing
- Control Flow Diagram
- Petri Nets
- Correctness Proof
- Scaling Analysis
- Table of Events
- Simulation
- 
-

## QUALITY ASSURANCE TECHNIQUES

- Software Inspections
- Software Reviews
- Software Audits
- Document Reviewed for
  - Consistency
  - Traceability
  - Clarity, readability
  - Structure
  - Completeness with respect to H/W & S/W
- Standards, Practices, and Conventions
- Configuration Management
- Quality Factors and Criteria
- Code Control and Media Control
- S/W Quality Assurance Standards
- 
- 

## IMPACT OF GOVERNMENT STANDARDS

MIL-STD-2167A  
MIL-STD-2168  
MIL-STD-F-9490D  
MIL-STD-483  
MIL-STD-490  
DI-S-30567A (CPDP)  
MIL-STD-1521A  
-



## GENERAL FACILITY DATA

General Data:	Company, laboratory name, address, key contacts, etc.
Computer Data:	Host processors, distributed processors, I/F diagrams, size of memory.
Cockpits:	Type (i.e., fighter, transport, generic) displays (analog, CRT, etc.) controls (i.e., stick, wheel, force-feel, etc.).
Graphics Data:	Resolutions, colors, update rates, features, etc.
Special Equipment:	As available from contractors containing data on special test support or software development equipment.
Software Library:	Program names and functions.
Facility Problems:	The contractor's estimation of areas of weaknesses and needed enhancements for flight critical software test.
Facility Plans and Goals:	New equipment expected, desired, or in procurement.
Processor Requirements:	What processors will be used to host in-house S/W development, memory requirements, timing data.
S/W Requirements:	Operating systems, languages (HOL and Assembly) FC Development tools for development test and demo. Emulators, etc. of all stages of development showing, S/W test and support requirements.
Contractor S/W:	Those software development, tests and demo packages that are required to support flight critical system development and verification including software availability from contractor with lease-rent-buy cost data.

## FLIGHT CONTROL SYSTEMS REQUIREMENTS ISSUES

- o Requirements. Methods should:
  - Be problem driven and applications based
  - Handle H/W and S/W interactions in concurrent mechanizations
  - Address complexity and reliability concerns explicitly and quantitatively
  - Effect component integration during design
  - Provide provision for quality assurance
  - Natural transition for system reqs to software design to H/W-S/W implementation and integration
- o Integrated support environment is needed.
- o S/W development environment will need to be incorporated into a system prototype facility.
- o Different levels of abstraction appropriate for different phase of development and post-development
- o MOST IMPORTANT REQUIREMENTS
  - o Satisfaction of fast real-time constraints
  - o High reliability
  - o High availability/survivability
  - o Supportability/ease of modification/ease of test

## GENERAL PROBLEM AREAS IN DEVELOPMENT

What measures of fault tolerant design are used?

What measures of fault avoidance design are used?

Structured design methods; QA; systematic testing; small, simple modules.

Methods of countering impact of:

- Increased use of relaxed static stability
- Integrated avionics and control functions
- Other: complex FCS Algorithms for sensor, sensor blending

## SYSTEM DEVELOPMENT ERRORS AND STATISTICS

Types of errors

Ease of finding

Measures of Performance

Gathering of statistics

## ERRORS

- (5) software integration testing and
- (6) software/system qualification

Errors in Tasks	Errors in Tasks			
	Stage 1 System Requirements	Stage 2 System Analysis and Design	Stage 3 Software Analysis and Design	Stage 4 Coding and Checkout Testing
	<p>System requirements are:</p> <ul style="list-style-type: none"> <li>• badly stated</li> <li>• changed from those written</li> <li>• incomplete or inconsistent</li> <li>• overly restrictive</li> </ul>	<p>System requirements are:</p> <ul style="list-style-type: none"> <li>• misinterpreted</li> <li>• poorly documented</li> </ul> <p>Data defining the system is incomplete</p> <p>Descriptions of the hardware and the peripheral equipment are inaccurate or incomplete</p>	<p>Lack of coordination between Stage 2 and Stage 3</p> <p>Poor documentation of the hardware</p> <p>Inadequate data on the configuration</p>	<p>Incomplete or erroneous design specifications</p> <p>Poor documentation of the hardware</p>
	<p>System requirements are:</p> <ul style="list-style-type: none"> <li>• misinterpreted</li> <li>• poorly documented</li> </ul> <p>Configuration of the system is not correct</p> <p>Size of the computer is inadequate</p> <p>Hardware interfaces are unclear</p> <p>Validation plan is not adequate</p>	<p>System configuration is wrong:</p> <ul style="list-style-type: none"> <li>• poor hardware/software trades</li> <li>• awkward executive control structures</li> <li>• faulty computer synchronizations</li> </ul> <p>Documentation of the specifications is incomplete</p> <p>Poor selection of computer language</p> <p>Software test procedures are poorly designed</p> <p>Concept review is done badly</p>	<p>Functions are partitioned incorrectly</p> <p>Software organization is insecure</p> <p>Poor utilization of:</p> <ul style="list-style-type: none"> <li>• software standards</li> <li>• guidelines</li> <li>• programming methodology</li> </ul> <p>Incorrect algorithms</p> <p>Documentation is incomplete and inadequate</p> <p>Preliminary design review badly done</p>	<p>Coding errors:</p> <ul style="list-style-type: none"> <li>• misinterpretation of language constructs and hardware operation</li> <li>• overflow, scaling and approximation</li> <li>• self-modification of instructions</li> <li>• sequencing and branching errors</li> <li>• incorrect loop structures</li> <li>• loss of index or state data</li> <li>• mishandling of singular and critical values</li> <li>• errors in computing equations</li> <li>• incorrect values for constants</li> <li>• unresizable code</li> <li>• uninitialized variables</li> </ul> <p>Code not reviewed</p> <p>Typographical errors</p> <p>Incomplete testing of modules</p> <p>Incorrect implementation of design specifications</p> <p>Poor documentation</p>

## REFERENCES

1. "LANGUAGE SUPPORT ENVIRONMENT FOR GUIDANCE AND CONTROL SYSTEMS", Final Report Working Group 08, NATO Advisory Group For Aerospace Research and Development, March 1990.
2. Fisher, Alan S., "CASE, USING SOFTWARE DEVELOPMENT TOOLS", John Wiley and Sons, 1988.
3. "VALIDATION OF FLIGHT CONTROL SYSTEMS", Draft Report of GCP Working Group 09, NATO Advisory Group For Aerospace Research and Development, April 1990.
4. "HANDBOOK - VOLUME I, VALIDATION OF DIGITAL SYSTEMS IN AVIONICS AND FLIGHT CONTROL APPLICATIONS", US DOT/FAA/CT-82/115, September 1986.
5. "DIGITAL SYSTEMS VALIDATION HANDBOOK - VOLUME II", US DOT/FAA/CT-88/10, February 1989.
6. "DIGITAL FLIGHT CONTROL SOFTWARE VALIDATION STUDY", AFFDL-TR-79 3076, June 1979.